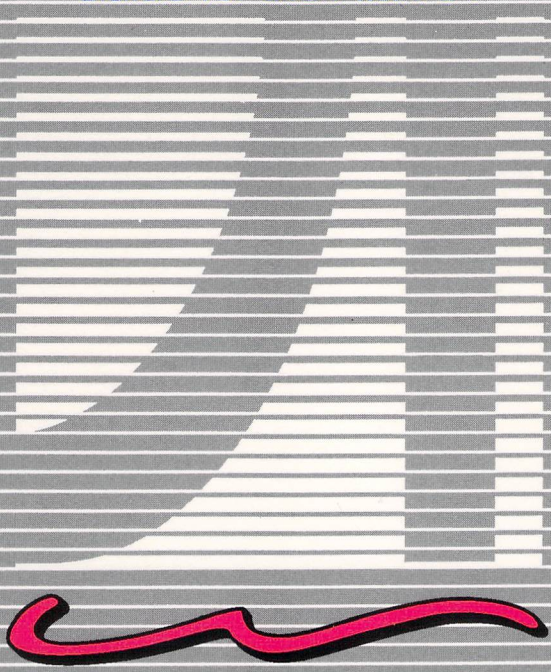
GFA BASIC

Quick Program Reference Guide for the Atari®ST™



Abacus
A Data Becker Book

GFA BASIC

Quick Program Reference Guide for the Atari ST

Uwe Litzkendorf
Michael Hösel

Abacus
A Data Becker Book

First Printing, February 1988

Printed in U.S.A.

ISBN 0-916439-90-9

Contents

Introduction

The **GfA BASIC Quick Reference Guide** is for the intermediate and advanced level programmer. However, it is not an introductory guide to the GfA BASIC language. We assume that you have a general knowledge of and some experience working with the Atari ST and GfA BASIC.

GfA BASIC sets new standards for BASIC. Using GfA BASIC it is possible to program serious BASIC applications with speed normally only available in a compiled language.

It is possible to use most GfA BASIC commands in direct mode (click the "direct" icon with the mouse or press the Esc key). Exceptions: Loop and GOTO commands, procedure definitions and status commands. Leaving direct mode and returning to the editor is done in one of three ways: Pressing the Esc key; typing ED, followed by the Return key; or by pressing the Control, Shift and Alternate keys at the same time.

GfA BASIC also allows the option of writing several routines in the editor and then calling these from direct mode. The GOSUB command is valid in this case and thus a procedure in the current program can be called. Therefore, leaving the interpreter can be avoided.

About this Guide

This guide has three major sections: introduction, general information, commands and three indexes.

The commands section describes all GfA BASIC commands in alphabetical order followed by the description and their syntax. In

addition to the clear organization of this guide, the syntax for commands have been presented in a concise form. The following characters have special meanings:

[] Many commands in GfA BASIC can be abbreviated. If a command can be shortened, the abbreviation is enclosed in braces. This information is then displayed both in the command title line and in all three indexes.

... If command options can be repeated they are indicated with ellipses.

The syntax of command descriptions is as follows:

All command names are in capital letters. Strings, variables or other parameters are displayed in lowercase letters.

Using the Indexes

If you know the name of a keyword, but are unsure of its usage or syntax, refer to the *Quick Index* at the back of this book. Each GfA BASIC command is listed in alphabetical order, and refers you to the page on which it can be found.

If you are looking for a keyword to perform a specific task, but are unsure of its name or syntax, refer to the Subject Index. There you'll find keywords listed in alphabetical order with short descriptions and the page on which it can be found.

For general information we have also included a General Index.

Sample GfA BASIC Command Page

Here's a quick explanation of the format this guide uses to describe GfA BASIC commands, descriptions and syntax.

Command **Brief description**

MENU	Create menu line

Syntax with parameters

```
MENU string_array()
```

Description

A string array is defined which contains the desired text for the menu titles and their menu options. `string_array()` is a one-dimensional string array which must have at least as many elements as menu titles and options are defined, plus an additional 20 elements for storing various organization strings.

The first menu has the following construction:

Parameter description

`String 1`
First menu title.

`String 2`
Any string. A program function can be assigned to this menu option. Since it is the only usable menu option in this menu, it is best suited for displaying program information.

`String 3`
Row of dashes. The number of dashes determines the width of the menu. Since this menu is used to access the desk accessories, the maximum accessory title length should be taken into account here.

`String 4 to String 9`
Six blank strings to serve as place holders for the accessories. No null strings ("") can be passed.

`String 10`
Null string ("") to mark the end.

General Information

You can enter most GfA BASIC commands in direct mode (click the direct field with the mouse or press the Esc key).

Exception: Loops and GOTO instructions, procedure definitions, and conditionals (IF...THEN).

You can return to the editor from direct mode by either pressing the Esc key or by typing the ED command, Return or simultaneously pressing the Control, Shift and Alternate keys.

The GOSUB command works in direct mode. You can use this to call a procedure in the current program. GfA BASIC allows you to write your own editor routines and then call them from direct mode. Under certain circumstances, this means that you can avoid leaving the interpreter. GfA BASIC offers commands which allow disk and printer operations external to the interpreter.

Since infinite loops are often used or encountered when programming in GfA BASIC, it should be mentioned that a program can be stopped by pressing the Control, Shift and Alternate keys simultaneously (see ON BREAK for exceptions).

The Editor

The following keys are used to position the cursor when using The Editor.

Cursor positioning

→ Move the editor cursor one position to the right in the current program line.

← Move the editor cursor one position to the left in the current program line.

Control + ←
Move the editor cursor to the start of the current line.

Control + →
Move the editor cursor to the end of the current line.

Tab Move the editor cursor to the right in steps of eight.

Control + Tab
Move the editor cursor to the left in steps of eight.

↑ Move the editor cursor one line above the current program line and check the syntax.

↓ or ↵ Move the editor cursor one line below the current program line and check the syntax.

Mouse (Position cursor at desired screen position and click)
Move the editor cursor on the current page to any location and check the syntax if the current line is left.

Clr/Home
Move the editor cursor to the start of the current editor page, check the syntax and structure.

Shift + F7 or PgUp from the menu
Move the editor cursor to the start of the previous editor page, check the syntax and structure.

Shift + F7 or PgDn from the menu
Move the editor cursor to the start of the next editor page, check the syntax and structure.

Control + Clr/Home
Move the editor cursor to the start of the program, check the syntax and structure.

Control + Z
Move the editor cursor to the end of the program, check the syntax and structure.

The Editor menu

Block operations

If a text block is defined with `BLK STA` and `BLK END`, the Block menu appears. If one of the two definitions is missing, "Block ???" appears.

Press F4 or select `BLOCK` from the Editor menu.

Change line height

The text editor can be displayed in 8 pixels with a maximum of 48 lines and 79 columns, or a character height of 16 pixels with a maximum of 23 lines and 79 columns.

Press Shift and F8 or select `TEXT 16` or `TEXT 8` from the Editor menu.

Change text input mode

Characters which you enter in the editor can have various effects on the existing program text. In the `INSERT` mode, the entered characters are inserted into the program at the cursor position. The rest of the line is moved to the right to make room for each character. In the `OVERWRITE` mode, the characters under the cursor are overwritten by the new characters.

Press F8 or select `INSERT` or `OVERWRITE` from the Editor menu.

Clear program
Answer Yes to delete the program currently in the editor, or No to return to the editor without changing anything.

Press Shift and F4 or select NEW from the Editor menu.

Copy block
A duplicate of the defined block is placed immediately before the cursor position. The original block is not changed.

Press C or select COPY from the Block menu.

Define block start
The line in which the cursor is located is defined to be the start of the block.

Press Shift and F5 or select BLK STA from the Editor menu.

Define end of block
The line immediately above the cursor is defined to be the end of the block.

Press F5 or select BLK END from the Editor menu.

Delete block
The defined block is immediately and irretrievably deleted (so be careful!).

Press Control and D simultaneously or select DELETE from the Block menu.

Display output window
The second interpreter window, used for output by graphics or text operations, is displayed. This process has no further effect on the editor. After pressing a key or clicking the mouse below the Editor menu, the interpreter returns to the editor.

Press F9 or select FLIP from the Editor menu.

Enable direct mode
The DIRECT or COMMAND mode is enabled.

Press Shift and F9 together, press Esc or select DIRECT from the Editor menu.

Exit the interpreter
You are asked if you really want to quit the program. If you answer Yes, you are returned to the desktop, otherwise you are returned to the interpreter.

Press Shift and F3 or select QUIT from the Editor menu.

Find and replace string
The editor waits for you to enter a character or string to be located in the program below the current cursor position. Then you must enter a string to replace occurrences of the first string found in the program.

Press Shift and F6 or select REPLACE from the Editor menu. After input, the search function can be called again by pressing Control and F. To replace the string found with the replacement string, press Control and R.

Find end of block
The cursor is placed immediately after the end of the block.

Press E or select END in the Block menu.

Find start of block
The cursor is positioned to the start of the block.

Press S or select START from the Block menu.

Find string
You are expected to enter a string which is located in the text. See above for more information. You can search for another occurrence of the same string with Control and F.

Press F6 or select FIND from the Editor menu.

List block
The defined block is printed. See Print program listing.

Press L or select LLIST from the Block menu.

Load program
A program with an extension of .BAS is loaded from disk. The loaded program replaces the current program in the editor. After a file has been selected from the file selector box, only a reset can stop the loading process. Therefore you must save any program before loading a new one.

Press the F1 key or select LOAD from the Editor menu.

Load text file
A program saved as a text file with the extension of .LST can be loaded from disk with this function. It also requires more time than the LOAD operation. It has the advantage, however, of combining various files with each other.

Press the F2 key or select MERGE from the Editor menu.

Move block
The defined block is moved to the location immediately before the cursor position. It is removed from its original position. The block definition is also removed.

Press M or select MOVE from the Block menu.

Page back
Press Shift and F7 or select PG UP from the Editor menu (see also the cursor functions).

Print program listing
Click Yes in the dialog box to print a formatted listing on the connected printer (Note: if a printer is not connected or if it is off line, the system waits about half a minute before returning to the interpreter. Don't press the Reset button!). Answer No to return to the interpreter.

Press F3 or click on LLIST in the Editor menu.

Save program
The current program is saved on disk with the extension of .BAS. This saves the program directly to disk as a complete memory block. This shortens the saving time considerably.

Press Shift and F1 simultaneously or select SAVE from the Editor menu.

Save text file
The program is saved as a text file on the disk with the extension of .LST. The time needed to save the program is considerably longer than with SAVE.

Press Shift and F2 together or select SAVE,A from the Editor menu.

Start program
The program in the editor is started.

Press Shift and F10 or select RUN from the Editor menu.

Test loop structure
The interpreter checks that all loops are terminated properly and returns to the editor.

Press F10 or select TEST from the Editor menu.

Write block
The defined block is saved to disk as a .LST file (by means of the file selector box). The block is not changed in the editor.

Press W or select WRITE from the Block menu.

Additional functions

Delete character to the left of the cursor
Press Backspace.

Delete character beneath the cursor
Press Delete.

Delete current cursor line
The line in which the cursor is currently positioned is deleted.

Press Control and Delete simultaneously.

Undo last change and restructure program
If a change to a program line is not confirmed by pressing Return, ↑ or ↓, the change can be undone by pressing the Undo key.

Insert blank line
The current cursor line is moved one line position down, and a blank line is inserted below the cursor. This line is then treated like any other new line. After entering new text and pressing Return, a new blank line is inserted until terminated by pressing either Return (without entering anything else on the line), the up- or down-arrow key or Undo.

Press the Insert key.

Disk Operations

With all disk operations where a filename is given, a search path can be defined where files can be found.

- Disk drive: Character followed by a colon indicating the disk drive. For example A: for the first drive and B: for the second...

- Question marks in the filename are for any single character. Asterisks represent an entire field (either before or after the period).

Examples:

B:\UTILITY*.*
Searches disk B in the UTILITY directory for all possible files.

B:\UTILITY\OUTPUTS*.HUH
Searches the subdirectory OUTPUTS for all files with the extension of HUH.

*.HUH Searches the current disk in the main directory or all files with the extension of HUH.

\DR???? Searches the current disks for all files which are seven characters long and begin with DR.

A:\DR????.HUH
Searches disk A for all files which are seven characters long, beginning with DR and ending with the suffix of HUH.

\D*.* Searches the current disk for all files that begin with the letter D.

Commands

!	Add comment

! comment text

Allows comments to be appended to the command line. The exclamation point has exactly the same function as the REM command. The difference is that the comment text which this symbol separates from the command text can be placed directly in the command line. This is not allowed for DATA and REM lines.

```
@S.input(24)                          ! hidden input
Print  A$                             ! output string
Procedure S.input(num%)               ! input routine
 Dpoke  Contrl+2,0                    ! no Ptsin's
 Dpoke  Contrl+6,2                    ! two Intin's
 Dpoke  Intin,4                       ! keyboard input
 Dpoke  Intin+2,1                     ! request mode
 Vdisys  33                           ! call Set_Mode
 Dpoke  Intin,num%                    ! number of characters
 Vdisys  31                           ! call input
 For I=0 To Dpeek(Contrl+8)           ! as often as char's
  A$=A$+Chr$(Peek(Intout+1+I*2))      ! build string
 Next  I                              ! end of loop
Return                                ! back  to  the program
```

This little routine allows hidden input (no echo) of strings of any length (num%). In principle it is identical to INPUT$. The input ends when the maximum number of characters is reached or when Return is pressed. The string is then copied into A$.

*

Designate variable pointer

```
*var
```

Placing an asterisk in front of a variable name causes the address of the variable in question to be returned. When a variable name var is designated as a pointer in this manner, the address of the variable, not its contents, is returned. For strings and arrays this is the descriptor, while for numerical variables it is the address at which the variable contents are located. VARPTR/ARRPRT can be used to access the variable indirectly.

ABS

Absolute value

```
ABS(x)
```

Returns the absolute value of a number. This is always greater than or equal to 0 (SIZE function). The ABS function returns the arguments given without a sign.

```
positive x (+x)        ABS(x) = x
x = 0                  ABS(x) = 0
negative x (-x)        ABS(x) = x
```

This function is often used in "financial" applications (There is no such thing as "negative money", rather debit amounts!) and in mathematical calculations (for example before the "radical sign" of unknown numbers).

ADD [AD] — Addition command

```
ADD v_name1, v_name2
ADD v_name1, constant
```

Addition of variables with result assignment. `v_name1` and `v_name2` can be numeric variables or array variables. `constant` is a numeric constant. The component following the comma is added. The result of the calculation is in the variable `v_name1`.

ADDRIN — AES - Address input block

Start address of a memory area for storing the addresses which are passed to an `AES` function.

If addresses are required by the function (such as object-tree addresses), they must be passed with `LPOKE` as integers in the `ADDRIN` array (four bytes per element).

ADDROUT — AES - Address output block

Start address of a memory area in which addresses returned by the AES function are stored.

If addresses are returned by an AES function, they can be read from the ADDROUT array with LPEEK (four bytes per element).

ALERT [A] — Create alert box

```
ALERT Icon%,B_text$,D_Button%,B_text$,Backvar%
```

Calls a function for limited development of user dialogs. The ALERT command consists of five components:

Icon%	0 = No symbol 1 = Exclamation point 2 = Question mark 3 = STOP sign
B_oxtext$	Here the actual text (message/question) is passed to the function. The | character is used as a separator between individual lines. A maximum of four lines with 30 characters each can be displayed. The text can be written directly in the command or passed as a string variable.
Def_Button%	The number of the default button is passed in this parameter. The default button is the one that is activated if the user presses the Return key. This button is drawn with a darker border (0=no default button).

`Buttontext$` Here the buttons are labeled. Each button can have a maximum of eight characters of text. The | character is also used here as the separator between individual buttons.

`Backvar%` The last parameter is a numerical variable in which the number of the button pressed (1, 2, 3) is passed from the function.

ARRAYFILL [AR] — Fill an array with a value

```
ARRAYFILL array(),x
```

Fills all elements of an array with a constant. `array` is the name of a desired numeric or boolean array variable. The array must be dimensioned. All elements of the array are set to the constant `x`. `x` can be either a number or a numeric variable. Filling string arrays is not possible. The type of `x` must be the same as the `array` type.

ARRPTR — Determine field descriptor address

```
ARRPTR(string_name$)
ARRPTR(array_name())
```

Returns the starting address of the string or array descriptor.

ASC — ASCII value ->text character

```
ASC("character")
```
(determine the ASCII value of the given character)

```
ASC("string")
```
(determine the ASCII value of the first character in the string)

```
ASC(string_variable)
```
(determine the ASCII value of the first character in the string variable)

The ASCII code of a text character can be determined. `ASC` is the inverse of `CHR$`. If the value of a single character should be determined, it is placed in quotes within parentheses. Similarly a multiple character string can be input, in which case the ASCII value of the first character in the string is returned. If a null string is input (""), the function returns to 0. The value from `ASC` can be directly output, placed in a numeric variable or used in a condition statement.

ATN — Return arctangent

```
ATN(x)
```

Calculates the arctangent of a value (arctangent = angle in radians for a given tangent value). The argument `x` is a tangent value, from which the angle is returned in radians. `ATN(x)` returns a value between -PI/2 and +PI/2. To convert this angle to degrees, multiply the result by 180/PI.

BASEPAGE — Return BASIC basepage

```
BASEPAGE
```

Returns the address of the `BASEPAGE` of the GfA BASIC interpreter. Every compiled or assembled program started from the desktop or with `EXEC` is assigned a base information block (`BASEPAGE`) of 256 bytes. The first 128 bytes contain basic data for the execution of the program. The remaining 128 bytes can be used in various ways. For example, if a command line was entered when the program was executed, it is found here, or the disk-transfer buffer (GEMDOS routine $1A) can be placed here.

BGET — Load part of a file into memory

```
BGET #channel,address,number
```

Reads part of a file into a specified memory address. `channel` is the identifier of a file opened with `OPEN`. From the file pointer position, `number` is how many bytes is read from a file into memory starting at `address`.

BIN$ Numeric -> Binary

```
BIN$(expression)
```

Converts a decimal, hexadecimal or octal value to the binary representation returned in a string. `expression` is a variable, a value or numerical expression in one of the other three number systems. The string returned with `BIN$` can be directly output with the `PRINT` command, placed in a string variable or used in a string expression. If a value is given in binary format, the prefix &X must be used (for example, &X10011101). The interpreter excepts binary values in this format for numbers represented in another system. Only integers can be represented in this format.

BIOS Call TOS routines

```
BIOS(opcode,parameter_list)
BIOS = Basic Input/Output System
```

The operating system routines offered by the various subsystems can be called with three commands, `BIOS`, `XBIOS` and `GEMDOS`. The Atari ST has three different system levels, each of which has special tasks. For example, disks can be formatted, font tables can be loaded, and so on by specifying the appropriate `opcode` (function number). The number of parameters passed depends on the system routine called. If long words are expected by the function (such as for address parameters), the corresponding value should be prefixed with L:. If no prefix is present, the passed values are interpreted as 16-bit values. If strings are passed, the start address must be specified (see also `VARPTR`). A numerical variable can be specified to receive the result of a parameter returned to the program.

BIOS
DRVMAP Determine connected drives

```
A%=Bios(10)
```

This function determines which drives are connected and available at the moment. The following values can be returned:

3	A and B are available
7	A, B and C are available
11	A, B and D are available
15	A, B, C and D are available

Even if no drives are connected or only drive A is connected, the value returned is still 3, since the function always assumes that at least A and B are available.

BIOS
GETBPB Get BIOS parameter block address

```
A%=Bios(7,Dev%)
For I=0 To 7
 Print Dpeek(A%+I*2)
Next I
```

The address of the 18-byte BIOS Parameter Block for the specified drive is returned. The designator for the desired drive is passed in `Dev%` (0=A; 1=B). Upon return, `A%` contains the BPB address. Here are the values stored in the parameter block (in order):

Size of sector in bytes
Clusters per sector
Size of cluster in bytes
Directory length in sectors

Size of a FAT in sectors
Sector number of 2nd FAT
Sector number of 1st data cluster
Number of data clusters present

BIOS
GETMPB Return MPB address

```
A$=Space$(12)
Void Bios(0,L:Varptr(A$))
Mfl%=Lpeek(Varptr(A$))
Free.s%=Lpeek(Mfl%+4)
Free.l%=Lpeek(Mfl%+8)
```

This function allows you to determine the address of the Memory Parameter Block, which can then be analyzed. A 12-byte null string is set up in `A$` which stores the MPB on return. The structure of the block is not explained here. In this example, the start address of free memory is placed in `Free.s%` and its size is placed in `Free.l%`.

BIOS
GETREZ Determine screen resolution

```
A%=Xbios(4)
```

The current resolution of the screen is returned. The following values can occur:

0 = 320x200 pixels
1 = 640x200 pixels
2 = 640x400 pixels

BIOS
KBSHIFT Test/set modifier key status

```
A%=Bios(11,Key%)
```

Here the status of the keyboard modifier keys can be tested or set. A positive value in `Key%` is interpreted as a key status (see values below). If the value -1 is passed for `Key%`, the current status is returned. Key combinations are represented by adding the values. The following values can occur in `A%`:

1	Right Shift key pressed
2	Left Shift key pressed
4	Control key pressed
8	Alternate key pressed
16	Caps Lock on

BIOS
MEDIACH Check for disk change

```
A%=Bios(9,Dev%)
```

This function can be used to determine if the disk in the current drive has been changed. `Dev%` contains the designator of the drive to be checked (0=A; 1=B). Upon return, the following values can occur in `A%`:

0	Disk is not changed
1	Disk may have been changed
2	Disk is changed

BIOS
RWABS

Read/write disk sectors

```
A$=Space$(Num%*512)
Void Bios(4,Rwf%,L:Varptr(A$),Num%,Rec%,Dev%)
```

With this function disk sectors can be read or written. A$ is used to set up a buffer to store the data read or to be written. This buffer must be 512 times as large as the number of sectors to be read or written (1 sector = 512 bytes). Rwf% determines the read (0) or write mode (1). Num% contains the number of consecutive sectors to be read/written. Rec% determines the sector at which to start (1-720) and Dev% contains the drive designator (0=A; 1=B; 2=hard disk).

If sectors are read, data can be displayed with PRINT A$.

BITBLT

Combine memory areas

```
BITBLT Q.db%(),D.db%(),R.db%()
```

This command combines a source raster with a destination raster in various modes (AND, OR, XOR, etc.). This allows direct access to a complicated VDI function (COPY RASTER). The most common use of this is to move areas on the screen or in memory (see GET/PUT). The method in which the copied raster operates on the destination area can be determined (see PUT). Three integer arrays must be prepared before the function is called. The first describes the source raster in six parameters (S.db%=source definition block), while the second array contains the same six parameters for the destination raster (D.db%=destination definition block). The two rectangles (screen area or idealized memory rectangle) are defined in points or bits by nine parameters in the third array (R.db%=rectangle definition block).

```
Q.db%(0) = Raster address
           (must be an even number).
      (1) = Raster width in points/bits
           (must be divisible by 16).
      (2) = Raster height in points/word lines
           (number of lines with the width
           defined above, which, when aligned
           vertically gives the height of
           the raster block)
      (3) = Raster width in words
           (always raster width in points/16)
      (4) = Raster format
           (always 0, because always device-
              specific)
      (5) = Number of raster planes (bit planes)
           (640/400=1 ; 640/200=2 ; 320/200=4)

Z.db%()  = Destination raster (see above)
R.db%(0) = X-coordinate of the upper left corner
              of the source rectangle (in points/bits)
      (1) = Y-coordinate der linken, oberen Ecke
              of the source rectangle (in points/word
              lines)
      (2) = X-coordinate der rechten, unteren Ecke
              of the source rectangle (in points/bits)
      (3) = Y-coordinate der rechten, unteren Ecke
              of the source rectangle (in points/word
              lines)
(4)to(7) = corresponding info for the destination
              rectangle.
      (8) = Combination mode (see PUT)
```

The two rectangles should have the same size in order to yield a predictable result. If the rasters are defined with different sizes, the combination is always done using the size of the source raster (careful!).

For experimenting it is advisable to limit operations to well defined areas of memory, because it isn't always easy to predict the result of the raster function.

BLOAD [BL] — Loads memory

```
BLOAD "Filename", Start
```

A data block which was saved with BSAVE is loaded back into memory. Filename is the name of the file to load. The memory location where the file should be loaded must also be given. A block length is not needed.

BMOVE — Move memory block

```
BMOVE source,destination,number
```

A given memory range is moved to another area. This very important and fast command allows large memory manipulations. In combination with the SPUT command this can be used for screen animation or scrolling. The memory block starting at source is copied into memory starting at destination. A total number of bytes is moved.

BOX, PBOX [B, PB] — Display a rectangle

BOX x_left, y_top, x_right, y_bottom
(empty rectangle)

PBOX x_left, y_top, x_right, y_bottom
(filled in rectangle)

Displays a rectangle, either empty or filled in with a pattern. The command must be given in two coordinate pairs (x_left/y_top and x_right/y_bottom). These are the opposite diagonal corners of the rectangle.

BPUT — Store memory into file

BPUT #channel,address,number

A defined area of memory can be inserted into an existing file. This function is the complement of BGET. An area of memory starting at address is written to a file specified by the identifier channel (from the OPEN command used to open it) at the file pointer position of the file. number bytes is moved.

BSAVE [BS] Saves memory

```
BSAVE "Filename", Start, Count
```

BSAVE allows a number of bytes to be saved from RAM onto the diskette. The `Filename` is the name that the block of memory should be saved under. After the comma, `Start` is the beginning RAM address at which the save begins. Following another comma, `Count` is the total number of bytes to save.

C: Machine language program (C-compiled) call

```
C:v_name(parameter_list)
```

A machine language program (compiled from C) can be called. `v_name` contains the address of the machine language routine. A `parameter_list` can be placed in parentheses. If no parameter list is given, use empty parentheses (). If 32-bit parameters are passed, the abbreviation L: is used. Without this specifier, parameters are assumed to be words. When returning to BASIC, the contents of register D0 can contain any desired long word. This value can be accessed with the `C:` function call and can be directly output, placed in a variable or used in a condition statement.

```
        A%=C:v_name()
or      PRINT C:v_name()
or      IF C:v_name()=value
```

CALL [CA] Machine language program call

```
CALL v_name
CALL v_name(parameter_list)
```

CALL enables you to call an assembled machine language program. v_name contains the address of the routine to call. If necessary, a parameter_list can be passed to the routine.

CHAIN [CH] Load program (Autostart)

```
CHAIN "Program name"
```

CHAIN loads a basic program from the disk and automatically starts it. It allows you to change programs as desired. When the new program is loaded, the BASIC memory area (current program) and the variable memory are both erased. If Program name doesn't have an extension, the interpreter uses the extension of .BAS. Program name can be used according to the file system hierarchy.

CHDIR [CHD] Change directories

```
CHDIR "Directory_name"
```

CHDIR makes it possible to change the current directory for disk operations. If just the new Directory_name is given, the disk operations take place in the new directory. Specifying a search path (\directory\subdirectory) according to the file system hierarchy a complete directory-subdirectory path can be set. If a deeper sub-directory is desired, the backslash (\) must be used. To go up the hierarchy structure, the backslash must be before the search path. If the Directory_name is simply the baskslash, the main directory becomes the current directory. CHDIR works on the current disk drive, changing drives is not possible (to do this see CHDRIVE). The quotation marks can be left off Directory_name.

CHDRIVE [CHDR] Set the current disk drive

```
CHDRIVE disk_drive
```

If a disk access command doesn't include a drive specification, then the current disk drive can be set with CHDRIVE. disk_drive is a value between 1 and 15 (1=Drive A, 2=Drive B,... 15=Drive O).

CHR$ ASCII -> text character

```
CHR$(value)
```

A string is returned consisting of the character for the given ASCII value. `CHR$` is the inverse of `ASC`. The first 32 ASCII values (0-31) are control characters and cannot be displayed with `CHR$`. There are a total of 256 ASCII characters (0-255), including control characters. If a value larger than 255 is input, the `value MOD 256` characters is returned. The result of `CHR$` can be directly output, placed in a string variable or used in a text expression.

CIRCLE, PCIRCLE [C, PC] Display a circle

```
CIRCLE x_pos, y_pos, radius
```
(complete circle)

```
CIRCLE x_pos, y_pos, radius
```
(complete filled in circle)

```
CIRCLE x_pos, y_pos, radius, alpha, beta
```
(portion of a circle)

```
PCIRCLE x_pos, y_pos, radius, alpha, beta
```
(filled in portion)

Displays a circle or a portion of a circle, either empty or filled in with a pattern. The coordinate pair `x_pos/y_pos` sets the center of the circle. `radius` sets the distance from the center to the edge of the circle. If a beginning angle `alpha` and an ending angle `beta` are at the end of the parameter list, a portion of a circle ("pie") is displayed. If `alpha` for example = 0, the portion begins

to the right of the midpoint on the X-axis (through the mid point). The ending angle `beta` follows counter-clockwise and is the ending point of the portion. Both angle inputs are in 1/10 of a degree (e.g. 90 degrees = 900).

CLEAR [CLE] — Clear fields and variables

```
CLEAR
```

All numeric variables are set to 0, character strings are set to null string. All arrays are cleared and their dimensions removed. The `CLEAR` command cannot be used in procedures or `FOR...NEXT` loops, since the index variable or the return pointer is erased. `CLEAR` is not necessary at the beginning of a program since all variables are erased with the `RUN` command.

CLEARW [CLE W] — Clear window contents

```
CLEARW handle
```

Clears the contents of a GEM window. The window identifier `handle` determines the number of the window (0...4) to be cleared. If it is the current window, the `CLS` command can also be used.

CLOSE [CL] — Close a data channel

```
CLOSE
```
(close all open files)

```
CLOSE #Channel_number
```
(close the file number `#Channel_number`)

`CLOSE` closes either a single file or all previously `OPEN`ed files. `Channel_number` is the file indicator (0-99) of the file to be closed. If the command is used without any parameters, all open files are closed. All files are automatically closed when `END`, `EDIT`, `SYSTEM` or `QUIT` are executed.

CLOSEW [CL W] — Close window

```
CLOSEW handle
```

Closes a GEM window. The window identifier `handle` determines the number of the window (0...4) to be closed.

CLR — Clear scalar variables

```
CLR var, var%, var$,...
```

Clears scalar variables. The name of the variable to be cleared or a list of variables to be cleared is passed to the command. This

command replaces the corresponding zero or null-string assignments.

CLS — Erase screen

```
CLS
CLS #Channel_number
```

When the command is used without parameters, the monitor or the opened GEM window is erased, and the cursor is placed in the upper left hand corner. Using CLS# output can be directed to either a diskette file or a virtual file. When the CLS# is used for a diskette file, the screen is cleared when the file is read.

COLOR [CO] — Set line color

```
COLOR color
```

Sets the color for line and point graphic commands. Each resolution has a certain number of colors that can be displayed. The current color for the commands BOX, RBOX, CIRCLE, DRAW, ELLIPSE, LINE and PLOT is set with the parameter COLOR from one of the possibilities listed below:

High Resolution	COLOR	= 0 or 1
Medium Resolution	COLOR	= 0, 1, 2 or 3
Low Resolution	COLOR	= 0, 1, 2,...15

With color systems, the color word for the COLOR command can be defined with the SETCOLOR command.

Pattern filling graphic commands such as PBOX, PRBOX, PCIRCLE and PELIPSE cannot be controlled with COLOR. Instead, see DEFFILL.

CONT [CON] Continue program execution after STOP

CONT

If program execution is halted with the STOP command, the input of CONT in direct mode can continue execution of the program at the line following the STOP command. CONT cannot be used after a CLEAR command. Also, CONT cannot be used if the program listing has been changed or new variables have been added.

CONTRL VDI - Control block

Start address of a memory area in which the basic parameters for VDI functions are stored.

VDI initialization parameters must be passed in the CONTRL array with DPOKE as integers (two bytes per element).

Contrl	Function number (opcode)
Contrl+2	Number of elements in the PTSIN array which should be read by the function.
Contrl+4	Contains the number of PTSOUT elements after the function terminates.
Contrl+6	Number of elements in the INTIN array which should be read by the function.

Contrl+8	Contains the number of INTOUT elements after the function terminates.
Contrl+10	Function specified identifier.
Contrl+12	Selected device handle.

COS — Cosine

COS(x)

Calculates the cosine of an angle given in radians. The cosine of an angle is defined for right triangles as the length of the adjacent side divided by the length of the hypotenuse. The x argument is an angle expressed in radians for which the cosine is calculated. If this value is given in degrees, it must first be multiplied by PI/180.

CRSCOL — Return screen cursor column

CRSCOL

This function returns the current cursor column relative to the screen. In contrast to the POS function, which returns the string relative cursor column, this command returns the column of the cursor on the screen (1-80). In combination with CRSLIN, this function forms the complement of PRINT AT(C,L).

CRSLIN — Return screen cursor line

```
CRSLIN
```

Returns the current screen relative cursor line. The TOS screen has a matrix of 80 columns by 25 lines using the 8x16 font. This function can be used to return the screen line in which the cursor is currently located (1-25). In combination with `CRSCOL` this function forms the complement of `PRINT AT(C,L)`.

CVI, CVL, CVS, CVF, CVD — Format numbers

```
CVI(2 character_string_expression)
CVL(4 character_string_expression)
CVS(4 character_string_expression)
CVF(6 character_string_expression)
CVD(8 character_string_expression)
```

A character string is converted into the appropriate number format. The string expression is converted to a single number in the given format.

`CVI` converts a 2 character string expression into a 16 bit integer value.

`CVL` converts a 4 character string expression into a 32 bit integer value.

`CVS` converts a 4 character string expression into a Atari BASIC real number.

CVF converts a 6 character string expression into a GfA BASIC real number.

CVD converts a 8 character string expression into a BASIC real number.

`string_expression` can be input directly as text or as a string variable. The returned function value can be directly output, placed in a variable or used in a condition statement. These functions are inverses of `MKI$`, `MKL$`, `MKS$`, `MKF$` and `MKD$`.

DATA [D] Save data

```
DATA item1, item2, "text_item1", text_item2...
```

The statement is a list of values or text separated by commas. Using the `READ` command, the given data can be read into variables. The text must not contain any commas. Enclosing the string in quotation marks is optional. In this case the interpreter reads all characters (including spaces) which are between the two commas. It is unimportant for the interpreter when reading alphanumeric characters (strings) if it finds numerical input. It is a different case however with a numeric `READ`. It must know what type of data is being read. The values can be in binary, hexidecimal or octal. Variables cannot be given in DATA statements. Comments in programs without any read statements can also be placed in a DATA line. This text is then a REM statement and is unimportant to the program.

DATE$ — Return system data

DATE$

Returns a string containing the current system date. DATE$ can be printed directly, assigned to a string variable, used in a string expression or a conditional statement. The output follows the format dd/mm/yyyy. If no changes are made to the current date, DATE$ always contains the current TOS version date.

DEC — Decrement (-1)

DEC v_name

Decrease the value of a numerical variable or array variable v_name by 1.

DEFFILL [DEFF] — Set fill pattern

DEFFILL color, fill_style, pattern
(choose pattern)

DEFFILL color, f_pattern$
(define pattern)

Sets the used fill pattern and allows a pattern to be defined. The pattern is used as a fill-in with graphic commands like FILL, PBOX, PCIRCLE, PELLIPSE and PRBOX. These all require the

input of a current fill pattern. Using the parameters `fill style` and `pattern`, 36 operating system patterns can be chosen.

Fill style	Pattern
0 = background (full)	omitted
1 = object color (full)	omitted
2 = points	1 to 24
3 = lines	1 to 12
4 = user defined	"ATARI" or own

With the `DEFFILL` variation `COLOR`, `f_pattern$` a desired pattern can be defined. Similarly as with the mouse pointer definition a 32 byte string is set in the string variable `f_pattern$`. The fill pattern is constructed from a 16*16 array of pixels. Only 16 words need to be given, the row after the bit pattern contains the previous fill pattern. This works best with the `MKI$` function.

```
Restore F.illdatas
For I%=1 To 16
      Read F.line%
      F.pattern$=F.pattern$+Mki$(F.line%)
Next I%
Deffill ,1,1
Pbox 100,100,320,200
Deffill 1,0
Pbox 320,100,540,200
Deffill 1,4
Pbox 100,200,320,300
Deffill 1,F.pattern$
Pbox 320,200,540,300
For I=1 To 25
      Deffill ,2,I
      Pbox I*23,0,I*23+23,40
      Deffill ,3,I Mod 13
      Pbox I*23,360,I*23+23,399
Next I
U=Inp(2)
Edit
F.illdatas:
Data &X0000000000111000
```

```
Data &X0000000001010000
Data &X0001111110111000
Data &X0010011101000100
Data &X0100001110000010
Data &X1010000000000010
Data &X1100000000000001
Data &X1010000000000001
Data &X1101000000000001
Data &X1010100000001011
Data &X0101010101010100
Data &X0010101010101000
Data &X0001010101010000
Data &X0000111010110000
Data &X0000000000000000
Data &X0000000000000000
```

DEFFN — Define a function

```
DEFFN function_name=function_expression
DEFFN function_name(variable_list)=
function_expression
```

Mathematical or alphanumeric functions can be compactly defined. `function_name` is any desired name for the function to be used with the `FN` command. The function name has the result of the function expression, as given in `function_expression`. The function name can be anything except a BASIC command name or a variable name which is already used. The first character can also be a digit. With the optional parameter `variable_list`, several variables can be input, which are then ordered with `FN`. This variable list can contain variables of many different types, as long as the proper order is maintained in the `FN` call. In the function definition, the only operations allowed are those that are the same type as the function definition. For example with `DEFFN FUN$`, only string operations are possible.

When calling the function, the current contents of the variables are used. The maximum length of a function is 256 characters. If this is too short, another function can be called directly after the first. Functions can be defined anyplace desired in the program. At the start of the program, all DEFFN variables are initialized. This can also happen at the end of the program when the FN function call is executed. An "endless loop" where two functions call each other cannot be broken out of with the Control/Shift/Alternate sequence. The same is true of recurssive function calls.

DEFLINE [DE] — Set line mode

```
DEFLINE l_style, l_thickness, b_form, e_form
```

Set the appearance of the lines which are used with the commands BOX, CIRCLE, LINE, RBOX, ELLIPSE and DRAW. Four parameters can be input:

`l_style`

0 = Line in background color
1 = Continuous line
2 = Broken line (short distance)
3 = Points
4 = Point - line - point
5 = Broken line (large distance)
6 = Line - point - point
-1 to -32767 = User defined line

The user defined line style is a 16 bit word in which a set bit is a point in the line and a unset bit is a blank pixel. This value must be given as a negative value in the first parameter. The line then appears as described by this 16 bit value. Displaying various line forms is only possible when using a `l_thickness` which has a value of 1.

`l_thickness`

The second parameter sets the thickness of the line. A maximum thickness of 40 pixels is possible. The thickness should be incremented in steps of 2.

`b/e form`

The form of the line beginning and ending is set with these three parameters.

0 = square
1 = arrow
2 = round

DEFLIST [DEFLIS] Set listing format

`DEFLIST X`

`DEFLIST` sets the appearance of the program listings. The GfA BASIC Editor can display the program in two different fashions:

`x <> 0` All beginning letters of BASIC commands and variable names are capitalized. (`Print, A, B, Variable, Left$("Text")`)

`x = 0` All BASIC commands are capitalized. All variable names are lowercase. (`PRINT, a, b, variable, LEFT$("text")`). The input parameter `x` sets the listing. The first display mode is activated when the interpreter begins. This command can only be used in direct mode.

DEFMARK [DEFM] Set marker symbol

```
DEFMARK marker_color,marker_type,marker_size
```

The color, type and size of the marking symbol used in POLYMARK is set.

type 1 = a single pixel point
type 2 = a plus sign
type 3 = a asterisks
type 4 = a rectangle
type 5 = a diagonal cross (X)
type 6 = a diamond

All markers are set at the line thickness of a pixel. The size can be changed in steps of 20.

DEFMOUSE [DEFMO] Set mouse form

```
DEFMOUSE form
DEFMOUSE mouse$
```

Set a user defined or system mouse form.

Mouse forms (system defined):

form = 0 = arrow
form = 1 = double brace (][)
form = 2 = bee
form = 3 = pointing hand
form = 4 = open hand
form = 5 = fine sights
form = 6 = thick sights
form = 7 = outlined sights

Instead of the `form` value, a string variable can be input to define the mouse form. In this case, all data are 2 byte words in `MKI$` format. The mouse is 16 lines by 16 columns.

Word 1 x-coordinate of the action point in the mouse form.

Word 2 y-coordinate of the action point in the mouse form.

All mouse action (for example MOUSEX, MOUSEY) is carried out at the action point.

```
MOUSE$ = MOUSE$ + MKI$(1) + MKI$(1)                 = upper left
MOUSE$ = MOUSE$ + MKI$(1) + MKI$(65535)             = lower left
MOUSE$ = MOUSE$ + MKI$(65535) + MKI$(1)             = upper right
MOUSE$ = MOUSE$ + MKI$(65535) + MKI$(65535)         = lower right
```

Word 3 always MKI$(1)

Word 4 color (White = MKI$(0), black = MKI$(1))

Word 5 cursor color (mouse)

Word 6 to 21
mouse mask pattern

Word 22 to 37
mouse picture pattern

```
Restore M_data.mask
For I%=0 To 15
     Read Data%
     M_mask$=M_mask$+Mki$(Data%)
Next I%
Restore M_data.form
For I%=0 To 15
     Read Data%
     M_form$=M_form$+Mki$(Data%)
Next I%
M_data$=Mki$(1)+Mki$(65535)+Mki$(1)+Mki$(0)+Mki$(1)

M_data$=M_data$+M_mask$+M_form$
Defmouse M_data$
Deffill ,2,8
Pbox 100,100,540,300
Repeat
```

```
      Mouse X,Y,K
      Plot X,Y
Until Mousek
Edit
M_data.mask:
Data 112,248,508,1022,2047,4095,8191,8190,16380
Data 16376,32752,32736,65472,65280,64512,61440
M_data.form:
Data 32,80,168,332,658,1317,2634,3220,5416
Data 4688,8608,8384,21248,19456,61440,49152
```

DEFNUM — Round output values

```
DEFNUM place
```

All of the values which are output after this command are executed and rounded to the specified place (3 to 11). For real numbers, the digits after the defined rounding place are output as zeros. If the `place` is after the decimal, all digits after it are ignored. The rounding is performed with `INT(A+0.5)`. If variables are printed this way, the contents are not affected by the rounding. The rounding affects all numerical outputs until it is changed with the next definition or turned off (`DEFNUM 11`).

DEFTEXT [DEFT] — Set graphic text mode

```
DEFTEXT color,type,rotation,size
```

The various appearances of the output characters can be set. Outside of a GEM window this effects only characters output with

`text`. Inside of a window (HANDLE 0 to 5) the characters output with `PRINT` can also be changed.

The parameters have the following meaning:

`color` Set which register the text color should be taken from. `type` sets the text form. There are 31 value forms:

value 0 = normal
value 1 = bold
value 2 = light
value 4 = italic
value 8 = underlined
value 16 = outlined
value 29 = italic outlined, bold, underlined.

`rotation` Set the text rotation angle

value 0	= normal
value 900	= text output sideways from bottom to top
value 1800	= text output upside down from right to left
value 2700	= text output sideways from top to bottom

`size` Set the text height (Values of 0 to 26):

value 0 = nothing
value 4 = desktop icon size
value 6 = color (8*8)
value 13 = normal (8*16)
value 26 = enlarged

DFREE — Returns the amount of free disk space

```
DFREE (disk drive)
```

Returns how much free space is available on the given diskette. The value returns the number of free bytes. `disk drive` is the number of the desired disk drive.

```
DFREE (0)
```
(Current disk drive)

```
DFREE (1...15)
```
(Drive A...O)

DIM [DI] — Dimension an array

```
DIM v_name (index1, index2, index3...)
DIM v_name1 (indices), v_name2 (indices)...
```

Sets the dimension of one or more arrays and reserves memory for them. `indices` is a list of integer values or numeric variables. The command initializes a desired array with the name `v_name`, `v_name1...`, in which the number of `indices` indicates how many dimensions the array should have (for example two-dimensional = Matrix). The `index` itself tells how many elements are in each dimension and is also the upper index limit (smallest index = 0). With multi-dimensional arrays the maximum number of elements is 65535. Single dimensional arrays are limited only by the available memory. The dimension of an array can only be erased with the ERASE command.

DIM? Determine the size of an array

```
DIM?(array())
```

Returns the number of elements in a desired array. `array` is the name of a desired array variable. If the array has not been previously dimensioned, a value of zero is returned. Important: The indexing of every array begins with zero.

DIR Output a directory

```
DIR
DIR "file selection criteria"
DIR "file selection criteria" TO "filename"
DIR "file selection criteria" TO "LST:"
```

`DIR` returns the contents of a diskette or a disk directory. The file listing can be displayed on the monitor, a disk file or a virtual file (for example `CON:` for console) by using the optional parameter `TO`. Output to the printer is accomplished with the `LST:` option (also see `OPEN`).

If a disk drive is not specified, the contents of the current drive are listed.

`DIR "\*.*"` Lists all files in the directory of the current disk on the monitor.

`DIR "B:\directory" TO "VID:"`
Lists all files in the `directory` directory on disk drive B to the virtual file "`VID:`" (that is the monitor without control characters).

DIR$ — Returns the current directory name

```
DIR$(disk_drive)
```

DIR$ returns the name of the last opened directory. disk_drive stands for the the desired disk drive and is one of the following:

DISK DRIVE=	0 - current disk drive
DISK DRIVE=	1...15 - Disk Drive A...O

The current directory name can be printed as (PRINT DIR$(0)) or placed in a string variable (A$=DIR$(0)). If there is no open directory, a null string is returned.

DIV — Division command

```
DIV v_name1, v_name2
DIV v_name1, const
```

Division of variables with result assignment. v_name1 and v_name2 can be numeric variables or array variables. const is a numeric constant. The value before the comma is divided by the value after the comma and the result is placed in the v_name1 variable.

DO...LOOP [DO...L] Endless loop

```
DO
...program commands...
LOOP
```

A DO...LOOP loop can only be broken out if a END command (END, EDIT, STOP) or a EXIT IF command with the condition true or a GOTO command inside of the loop to a label outside of the loop is encountered. It is also possible to exit the loop with the GOSUB command to another procedure and then back into the loop. Otherwise, the loop can be terminated by using the break sequence Control/Shift/Alternate. DO...LOOP loops can be nested as deeply as desired.

DRAW [DR] Connect points with a line

```
DRAW x_pos, y_pos
```
(display 1 point)

```
DRAW TO x_pos, y_pos
```
(draw to a point)

```
DRAW X1, Y1, TO X2. Y2...TO Xn, Yn
```

Displays a graphic point and connects this with a line. Giving a coordinate pair x_pos/y_pos with DRAW displays one graphic point. In this case DRAW is identical to the PLOT command. The optional parameter TO indicates that the current point is connected with a line to the given point (can also be the endpoint of a LINE command). If the TO option is used several times, any desired figure can be displayed.

EDIT [ED] — End program

```
EDIT
```

EDIT causes the program to end, erases all variables and closes all open files. This command is identical to the END command with the exception that the GfA BASIC interpreter doesn't display an end of program window, rather it returns directly to the editor. The QUIT command completely exits the program. It cannot be restarted with the CONT command. All open files are closed, and all program variables are erased. In direct mode EDIT returns you to the Editor.

ELLIPSE, PELLIPSE [ELL, PE] — Display an ellipse

```
ELLIPSE x_pos, y_pos, a, b
(ellipse)

PELLIPSE x_pos, y_pos, a, b
(filled in ellipse)

ELLIPSE x_pos, y_pos, a, b, alpha, beta
(portion of ellipse)

PELLIPSE x_pos, y_pos, a, b, alpha, beta
(filled in portion)
```

Displays an ellipse or a portion of an ellipse. An ellipse is defined along two axis. a defines an ellipse in the x direction, b in the y direction. The center of the ellipse is set with the coordinate pair x_pos/y_pos. If the values alpha and beta are at the end of the parameter list, a portion of the ellipse is displayed. alpha is

the beginning angle (0 degrees = right of center on X axis), and `beta` is the ending angle (counter clockwise). These values are given in 1/10 of a degree (900 = 90 degrees).

END — End program

```
END
```

The program is stopped as soon as the `END` command is executed. A program end window is displayed, after which the editor is reentered. This command completely exits the program. It cannot be restarted with the `CONT` command. All open files are closed, all program variables are erased.

EOF — Determine the end of a file

```
EOF (#Channel_number)
```

Checks if the last byte has been read from the open diskette file on `Channel_number`. This command is used to check for the end of a file. `Channel_number` is the identification number (0-99) of the file to be tested. If the file pointer is at the last byte of the file, the value -1 is returned, otherwise 0 is returned. This value can either be placed in a variable `(A=EOF(#1))` or used directly in a condition statement.

ERASE [ER] — Erase an array

```
ERASE array()
```

Erases an array and removes its dimensions. `array` is the name of the array variable to erase. The reserved memory is freed up and can be used for another application.

ERR — Return error code

```
ERR
```

After an error occurs, this variable contains an identification number which corresponds to the error. `ERR` is a reserved variable in which a number between -128 and 127 is stored when an error occurs. `ERR` can be printed directly, assigned to a numerical variable or used in a conditional.

ERROR [ERR] — Simulate error

```
ERROR error_number
```

Simulates an error with the specified `error_number`. An error message is printed, or if `ON ERROR GOSUB` was used, control branches to the error procedure specified. After an error message is printed, the program is stopped and the interpreter returns to the editor.

EVEN — Test for even number

```
EVEN(number)
```

The specified number is tested to see if it is even. The value -1 (true) is returned if the value is even. If it is not, the value 0 (false) is returned.

EXEC — Load and start .PRG/.TOS program

```
EXEC mode,"prg","cmd","env"
```
(as command)

```
A=EXEC(mode,"prg","cmd","env")
```
(as function)

.PRG/.TOS programs can be loaded and started. Almost all compiled or assembled programs of type .PRG or .TOS can be loaded. To do this, a sufficiently large memory space must be allocated (see RESERVE). mode contains either 0 (load and start) or 3 (load). "prg" is the complete pathname of the program. "cmd" contains a command line to be passed to the program (see BASEPAGE). In most cases a null string can be passed. "env" is a string which describes the program environment. This string is not sufficiently documented by Atari to be able to adequately judge its use. A null string can be passed.

If EXEC is used as a function, a value is returned by the called program if mode=0. If mode=3, the address of the BASEPAGE is returned after it is loaded.

Memory resident programs (SID), the SEKA assembler, and any programs which use any form of batch files cannot be executed in this manner.

```
@Executer
Procedure Executer
 Fileselect "*.*","",Selected$
 If Selected$>"" And Selected$<>"\"
  If Exist(Selected$)<>0
   S.pace=Fre(0)
   Reserve 50000
   Alert 2,"1. Load & start|2. Load &
    return basepage",1," 1. | 2.
",Dummy%
   If Dummy%=1
    Exec 0,Selected$,"",""
    Reserve Xbios(2)-Himem-16384+Fre(0)

   Else
    A%=Exec(3,Selected$,"","")
    Print A%
    Alert 2,"Release memory?",
     1," Yes |No",Dummy2%
    If Dummy2%=1
     Void Gemdos(&H49,L:A%)
     Reserve Xbios(2)-Himem-
16384+Fre(0)
    Endif
   Endif
  Endif
 Endif
Return
```

EXIST — Check for the existence of a file

```
EXIST(Filename)
```

EXIST determines if the input Filename is on the diskette. It returns either a 0 (not there) or a -1 (there). This value can be placed in a variable (A=EXIST(Filename)) or directly in a condition statement (IF EXISTS(Filename) <>0). Filename can also be a pathname.

EXIT IF [E IF] — Condition loop exit

```
EXIT IF condition
```

Exits a loop regardless of the loop exit condition, if the given condition is true. This command is a special function and is only valid inside of FOR/NEXT, DO/LOOP, REPEAT/UNTIL and WHILE/WEND loops. If the condition specified in the EXIT IF command is true, the loop is broken out of and execution continues at the next line after the loop end (NEXT, LOOP, UNTIL or WEND).

EXP — Base E exponentiation

```
EXP (x)
```

Calculates the value of the constant E raised to an exponent. The function uses E as the base and x can be any desired number. The constant E is called Euler's number (E=2.718281828...) and is the base of natural logarithms.

FALSE — False constant (0)

```
FALSE
```

Contains the constant 0. A boolean value is returned by various functions (such as EXIST), and for better readability, this constant can be used in conditions instead of the value 0 (see also TRUE).

FATAL — Return error type

```
FATAL
```

After an error occurs, this variable contains an identification number corresponding to the type of error which occurred. A distinction is made between normal errors and bomb producing errors. If a bomb error occurs (address of the last BASIC command executed is unknown), a value of -1 is returned. For all other

errors a 0 is returned. If an error procedure is defined with ON ERROR GOSUB, this value can be evaluated there, passed to a variable or printed. After an error message is printed, FATAL can be evaluated in the direct mode.

FIELD [FIE] Divides data records into fields

```
FIELD #Channel_number, length AS v_name1$,
length AS v_name2$...
```

FIELD divides a record into any number of given length fields. These fields then accept string variables. Strings shorter than the given field length are padded with spaces. Channel_number is the file that was opened as a "R" file (see OPEN) which should be divided into records. The OPEN command, "R", #Channel_number, "Filename", record_length sets the record length for a random access file. The sum of all field sizes must be the same size as the record_length value. If more than one field is desired, each field must be separated by commas. Every channel open in "R" mode must be followed by a FIELD definition. If more fields are specified than created, an error message appears. The FIELD command has a maximum of 256 characters for the total record length.

FILES [FILE] Output an (expanded) directory

```
FILES
```
(all files)

```
FILES "file selection criteria"
```
(monitor)

```
FILES "file selection criteria" TO "filename"
```
(disk listing)

```
FILES "file selection criteria" TO "LST:"
```
(printer listing)

FILES returns the contents of a diskette or a disk directory along with the file length (in bytes), date and time created. The file listing can be displayed on the monitor, a disk file or a virtual file (for example CON: for console) by using the optional parameter TO. Output to the printer is accomplished with the LST: option. If no drive is specified, the current disk drive is used.

FILESELECT [FILESE] Select file

```
FILESELECT "path","template",Backvar$
```

Creates a dialog box for file selection and returns the selected pathname. Three parameters are passed. The first is a string or string variable which contains the path which is displayed first. In the second a filename can be specified which is placed in the input line under selection when the box is displayed. It should be noted that a maximum of 12 characters (including periods) can be used (8 for filename / 1 for separator / 3 for extension). "" can also be used or just the extension of the file. The third parameter is a string variable in which the complete pathname of the file selected

by the user is placed. When the function returns, five different things can be returned in this variable. If a file was actually selected by the user with a double click on the appropriate name, or a single click on the name and then the OK box is selected, then its name is returned in the variable. However, if the OK box is clicked without a selection being made, there are three variations. If a filename is passed before placed in Selection and the OK box was used, then this filename is also passed back. Or, if only the extension is given, then the selected extension is in the variable. If no selection is made and no variable or extension is passed, or it is erased by the user and the OK box is clicked, then a backslash (\) is returned. If Cancel is clicked, the variable is completely empty ("").

FILL [FI] — Fill in a border with a pattern

```
FILL x_pos, y_pos
```

Fills a surrounded area with a fill pattern. The coordinate pair `x_pos/y_pos` sets the location where the fill should start. The fill pattern is set with the `DEFFILL` command.

FIX — Integer function

```
FIX(x)
```

Returns the whole (integer) portion of a real number by truncating the portion right of the decimal point. x is any desired numeric expression. The function rounds numbers neither up or down,

rather only removes the decimal portion. The fractional portion of a number is returned with FRAC. FIX is identical to TRUNC.

FN [@] — Function call

```
FN function_name
FN function_name(parameter_list)
```

FN calls a function defined with DEFFN. If a variable list is specified with DEFFN, the function call must also have a parameter_list, with parameters separated by commas. The variable types in the function definition and in the calling parameter list must match, otherwise an error window is displayed. The result of the function can either be directly displayed, placed in a variable of the function type or used in a condition statement.

FORM INPUT — Formatted input

```
FORM INPUT Number,V_name$
```

This command allows for string input of a determined length. The command is not automatically terminated, as INPUT$ is when the requested number of characters are input; rather the Return key must be pressed to quit. When the last position is reached, the bell is sounded. The maximum input length is 255 characters. Input can be corrected with Delete, Backspace and the arrow keys. The input of special characters is the same as described in the INPUT command.

FORM INPUT AS — Output string for editing

```
FORM INPUT number AS var$
```

Outputs the contents of a string variable on the screen and allows it to be edited. `number` contains the number of characters in the string `var$` to be outputted for editing. After the editing and confirmation with Return, the modified string is read into the variable again. The previous contents are completely replaced. If an empty variable is specified, this command has the same effect as `FORM INPUT number,var$`.

FOR...NEXT [F.N] — Counting loop

```
FOR count = start TO/DOWNTO end [STEP value]
...program commands...
NEXT count
```

Sets up a loop which is carried out as indicated in the loop control. The `count` variable is set to the value given in `START`. When the program encounters the appropriate `NEXT` command, `count` is moved up/down by the `STEP` amount. If `DOWNTO` and `STEP` are not given in the command, the `value` increment is +1. If `DOWNTO` is used without `STEP`, the increment is -1. `STEP` can be either a positive or negative value. `FOR...NEXT` loops can be nested as deep as desired as long as the `count` variables have different names. Whenever it is possible, the `count` variables should be integers. This allows for a shorter loop execution time, and integers use only 2/3 of the memory space as floating point numbers.

FRAC — Fraction function

```
FRAC(x)
```

Returns the decimal portion of a real number. The argument x can be any desired numerical expression. The function returns the decimal portion if x is a real number, 0 if x is a integer. FRAC is the complement to FIX.

FRE — Return free memory space

```
FRE(dummy)
```

Returns the size of the current free memory area in bytes. A "garbage collection" is performed and then the size of the free memory is returned. Garbage collection causes the working memory to be "cleaned up," whereby all unused areas of memory are collected and cleared. dummy is any integer value and has no meaning.

FULLW [FU] — Enlarge window to screen size

```
FULLW handle
```

Enlarges a GEM window to a full size screen, except for the menu bar. The window identifier handle determines the number of the window (1...4) to be enlarged to the X-/Y-coordinates 0/20,

639/20, 0/399, 639/399 (=window border points). The remaining area is reserved for the menu bar (see MENU for more).

GB — AES - Starting address

```
G.contrl% = Lpeek(Gb)
G.intin% = Lpeek(Gb+8)
G.intout% = Lpeek(Gb+12)
G.addrin% = Lpeek(Gb+16)
G.addrout%= Lpeek(Gb+20)
```

Start address of a memory area in which the start address of the AES arrays are stored. The locations listed above contain the same addresses as are available through the various BASIC variables with similar names.

GCONTRL — AES - Control block

Start address of a memory area in which the basic parameters for AES functions are stored.

AES initialization parameters must be passed in the GCONTRL array with DPOKE as integers (two bytes per element).

GCONTRL	Function number (opcode).
GCONTRL+2	Number of elements in the GINTIN array which should be read by the function.

GCONTRL+4	Contains the number of GINTOUT elements after the function terminates.
GCONTRL+6	Number of elements in the ADDRIN array which should be read by the function.
GCONTRL+8	Contains the number of ADDROUT elements after the function terminates.

GEMDOS — Call TOS routines

```
GEMDOS (opcode,parameter_list)
GEMDOS= Graphic Environment Manager Disk
Operating System
```

The operating system routines offered by the various subsystems can be called with three commands, BIOS, XBIOS and GEMDOS. The Atari ST has three different system levels, each of which has special tasks. For example, disks can be formatted, font tables can be loaded, and so on by specifying the appropriate opcode (function number). The number of parameters passed depends on the system routine called. If long words are expected by the function (such as for address parameters), the corresponding value should be prefixed with L:. If no prefix is present, the passed values are interpreted as 16-bit values. If strings are passed, the start address must be specified (see also VARPTR). A numerical variable can be specified to receive the result of a parameter returned to the program.

GEMDOS
Current Disk
Return current drive

```
A%=Gemdos(&H19)
```

Here you can determine which drive is the current drive. When the function returns, `A%` contains the drive designator.

Drive A=0; B=1; C=3; D=4; etc.

GEMDOS
Get DTA
Read disk buffer address

```
A%=Gemdos(&H2F)
For I=0 To 43
  A$=A$+Chr$(Peek(A%+I))
Next I
```

Yields the start address (DTA = Disk Transfer Address) of the 44-byte disk transfer buffer. This buffer is used by `Gemdos` disk functions to store various system data (such as the name of the file). When the function returns, the address is in `A%` and the contents of the buffer are in `A$`.

GEMDOS
Malloc
Reserve protected memory

```
A%=Gemdos(&H48,L:-1)
```
(Read memory size)

```
B%=Gemdos(&H48,L:A%)
```
(Reserve memory)

An area of memory protected from system access can be reserved with this function. In addition, it can also be used to read the available memory size. In the second call, `A%` contains the amount of memory to reserve and must always be less than or equal to the amount actually available. `B%` contains the start address of the reserved block (save for `MFREE`!). Note: The changes in operation with `RESERVE`.

GEMDOS
Mfree
Release reserved memory

```
Void Gemdos(&H49,L:Address%)
```

A memory block reserved with `Malloc` is released back to the system with this call. `Address%` is the start address of the memory to be released.

GEMDOS
Set Disk Transfer Address
Disk buffer address

```
Reserve Fre(0)-256
Void Gemdos(&H1A,L:Himem)
```

Sets a 44-byte memory area aside for the current DT buffer for `Gemdos` disk operations. In this case the DTA is set to `Himem`. When using search first, for example, the relevant data are stored in the new buffer. If necessary, it can be read as follows:

```
A$=Space$(44)
Bmove Himem,Varptr(A$),44
Print A$
```

GEMDOS
Set DRV
Set current drive

```
A%=Gemdos(&HE,Drive%)
```

This function is used to make a given drive the current one. `Drive%` is the designator of the desired drive. When the function returns, `A%` contains the designator of the last current drive.

Drive A=0; B=1; C=3; D=4; etc.

GEMDOS
Sfirst — Search for file

```
N.ame$+"File_name +chr$(0)
A%=Gemdos (&H4E,L:Varpt(n.ame$),&H0)
```

The directory is searched for a file with the specified name. Upon return, A% contains either a zero (function executed) or a negative value. All relevant data (filename, size, etc) are placed in the current disk transfer buffer. The file size can be read with SIE%=LPEEK (GEMDOS (&2F)+26).

GEMSYS — Call AES routines

GEMSYS
(repeat last opcode)

GEMSYS opcode
(new opcode)

AES system routines can be called with this command. The AES (Application Environment System) supports graphic oriented operations (windows, menus, alert boxes). opcode contains the function number of the GEM routine to be called. The GEM parameter arrays must be prepared before the call. If you just change the array contents and call the same function again, you can omit the opcode specification. The arrays to be prepared are ADDRIN, ADDROUT, GB, GCONTRL, GINTIN and GINTOUT.

```
For I=1 To 10
  Dpoke Gintin,620-I*55
  Dpoke Gintin+2,390-I*20
  Dpoke Gintin+4,20+I*20
  Dpoke Gintin+6,10+I*20
  Dpoke Gintin+8,10+I*20
  Dpoke Gintin+10,10+I*20
  Dpoke Gintin+12,620-I*20
  Dpoke Gintin+14,10+I*20
  Gemsys 73
  Gemsys 74
Next I
```

GEMSYS
APPL_READ Read event buffer

A given number of bytes are read from the application buffer.

```
Dpoke Gintin,Appl.ident
Dpoke Gintin+2,number
Lpoke Addrin,bufferaddress
Gemsys 11
```

GEMSYS
APPL_WRITE Write to event buffer

A given number of bytes is written to the application buffer.

```
Dpoke Gintin,ident. of the receiving application
Dpoke Gintin+2,number of bytes to write
Lpoke Addrin,address of the character buffer
Gemsys 12
```

GEMSYS
FORM_CENTER Center dialog box

The coordinates of the specified dialog box is adjusted so that it appears in the middle of the screen.

```
Lpoke Addrin,object true address
Gemsys 54
X.Coord%=Dpeek(Gintout+2)
Y.Coord%=Dpeek(Gintout+4)
Formwidth%=Dpeek(Gintout+6)
Formheight%=Dpeek(Gintout+8)
```

GEMSYS
FORM_DO AES control

AES monitors the dialog inputs and records termination conditions.

```
Dpoke Gintin,0 or index of the first text field
Lpoke Addrin,Objecttreeaddress
Gemsys 50
Endobject%=Dpeek(Gintout)
```

GEMSYS
GRAF_DRAGBOX — Move box in border

A screen area defined within a smaller box of a predefined size can be moved. This function is called when the mouse button is pressed. Last.x% and Last.y% contain the X/Y coordinates of the movable box when the mouse button is released.

```
Dpoke Gintin,width of the small box
Dpoke Gintin+2,height of the small box
Dpoke Gintin+4,X-start coord. (Mousex)
Dpoke Gintin+6,Y-start coord. (Mousey)
Dpoke Gintin+8,X-coord of the upper left corner
      of the enclosing rectangle
Dpoke Gintin+10,Y-coord of the upper left corner
      of the enclosing rectangle
Dpoke Gintin+12,width of the enclosing rectangle
Dpoke Gintin+14,height of the enclosing rectangle
Gemsys 71
Last.x%=Dpeek(Gintout+2)
Last.y%=Dpeek(Gintout+4)
```

GEMSYS
GRAF_GROWBOX — Expanding rectangle

A box is drawn which expands from a given size to another size in a different screen position.

```
Dpoke Gintin,X-coord. of the start rectangle
Dpoke Gintin+2,Y-coord. of the start rectangle
Dpoke Gintin+4,width of the start rectangle
Dpoke Gintin+6,height of the start rectangle
Dpoke Gintin+8,X-coord. of the destination rectangle
```

```
Dpoke Gintin+10,Y-coord. of the destination rectangle
Dpoke Gintin+12,width of the destination rectangle
Dpoke Gintin+14,height of the destination rectangle
Gemsys 73
```

GEMSYS
GRAF_HANDLE Return application handle

Returns the handle of the application making the call and the current character size.

```
Gemsys 77
Handle%=Dpeek(Gintout)
charwidth%=Dpeek(Gintout+2)
charheight%=Dpeek(Gintout+4)
charboxwidth%=Dpeek(Gintout+6)
charboxheight%=Dpeek(Gintout+8)
```

GEMSYS
GRAF_MKSTAT Return switch-key status

This function is used to determine if any of the switch keys Control/Shift/Alternate have been pressed. Combinations are returned as sums of the following codes:

Status 1	Right Shift key
Status 2	Left Shift key
Status 4	Control key
Status 8	Alternate key

```
Gemsys 79
Status=Dpeek(Gintout+8)
```

GEMSYS
GRAF_MOVEBOX Call MOVEBOX

A border of constant size is drawn which can be moved from one screen position to another in multiple steps.

```
Dpoke Gintin,width of the box
Dpoke Gintin+2,height of the box
Dpoke Gintin+4,X-start-coordinate
Dpoke Gintin+6,Y-start-coordinate
Dpoke Gintin+8,X-destination-coordinate
Dpoke Gintin+10,Y-destination-coordinate
Gemsys 72
```

GEMSYS
GRAF_RUBBERBOX Call rubberbox

A rubber box is a rectangular border where the upper left corner is fixed and the lower right corner follows the mouse pointer. The function is called when the mouse button is down. `Last.w%` and `Last.h%` contain the last width and height, repetitively, of the box in pixels.

```
Dpoke Gintin,X-coord of the upepr left corner
Dpoke Gintin+2,Y-coord of the upepr left corner
Dpoke Gintin+4,min. box width
Dpoke Gintin+6,min. box height
Gemsys 70
Last.w%=Dpeek(Gintout+2)
Last.h%=Dpeek(Gintout+4)
```

GEMSYS
GRAF_SHRINKBOX — Shrinking rectangle

A box is drawn at a given screen position which shrinks to a given size in a different screen position. The same eight parameters must be passed as described for GRAF_GROWBOX.

```
Gemsys 74
```

GEMSYS
MENU_TEXT — Change drop-down menu text

The text contents of a drop-down menu option can be changed while the program is running. The new string cannot be longer than the old one.

```
A$="new menu option text"+Chr$(0)
Dpoke Gintin,menu option index
Lpoke Addrin,menu tree address
Lpoke Addrin+4,Varptr(A$)
Gemsys 34
```

GEMSYS
OBJC_DRAW — Draw object tree

The object tree loaded with RSRC_LOAD is displayed on the screen within a clipping rectangle up to a specified depth.

```
Dpoke Gintin,index of the start object
Dpoke Gintin+2,number of levels to draw
Dpoke Gintin+4,X-coord of the clipping box
Dpoke Gintin+6,Y-coord of the clipping box
```

```
Dpoke Gintin+8,width of the clipping box
Dpoke Gintin+10,height of the clipping box
Lpoke Addrin,object tree address
Gemsys 42
```

GEMSYS
OBJC_OFFSET Return object coordinates

This function returns the screen relative (upper left corner of the screen/object) coordinates of an object. The data can be read from X.c%/Y.c% after the function terminates.

```
Dpoke Gintin,index of the object
Lpoke Addrin,object tree address
Gemsys 42
X.c%=Dpeek(Gintout)
Y.c%=Dpeek(Gintout+2)
```

GEMSYS
RSRC_FREE Release ".RSC" memory

The last area of memory used by RSRC_LOAD is released and the object tree is erased.

```
Gemsys 111
```

GEMSYS
RSRC_GADDR — Return object address

The address of the last loaded object tree or structure is returned.

```
Dpoke Gintin,0 (= tree) or structure type
Dpoke Gintin+2,Objectindex
Gemsys 112
Address%=Lpeek(Addrout)
```

GEMSYS
RSRC_LOAD — Load ".RSC" file

Reserves memory and loads a resource file in this area.

```
A$="Name of the resource file"+Chr$(0)
Lpoke Addrin,Varptr(A$)
Gemsys 110
```

GET — Save screen portion

```
GET X1,Y1,X2,Y2,v_name$
```

Reads a portion of the screen into a string variable. The upper left point (X1, Y1) and the lower right point (X2, Y2) define the range from which the pattern is read into the string variable. Using the PUT command, a desired portion of the screen can be output.

GET# Read a record

GET# Channel_number
(Read the next record)

GET# Channel_number, Record_number
(Read the given record)

Reads a record from an open random access file. Channel_number is the identifier (0-99) of the random access file which should be read (see OPEN). Every record in a random access file is numbered. Using the Record_number parameter, any desired record can be retrieved with GET#. If Record_number is not specified, the next record is read.

GINTIN AES - Integer input block

Start address of a memory block for storing function specific parameters passed to an AES function.

If the function must access parameters, they must be passed in the GINTIN array with DPOKE as integers (two bytes per element). The number of elements and their meanings depend on the function called.

GINTOUT AES - Integer output block

Start address of a memory block in which the function specific parameters are returned from an AES function.

If the function returns parameters, they can be read from the GINTOUT array with DPEEK as integers (two bytes per element). The number of elements and their meanings depend on the function called.

GOSUB [GO or @] Branch to a procedure

```
GOSUB Procedure_name
GOSUB Procedure_name(parameter_list)
```

A procedure can be branched to and a parameter list given with this call. Procedure_name is the name of the procedure which should be branched to. The optional parameter parameter_list can be as long as desired. It is important to note that the number of parameters in the call must be the same as the number in the procedure definition. Also, the data types must match. The data types can be mixed in any desired order, so long as the calling list has its data types in the same order.

GOTO [GOT] — Branch to a label

```
GOTO label
```

Using GOTO enables branching to any desired portion of the program. With GOTO there is no implicit return built in (as in GOSUB). The program continues execution from the named line. LABEL is any desired name which serves as a marker for GOTO or RESTORE (see RESTORE) commands. As with procedure names, it is also possible for the first character of a label to be a digit. Labels can be placed any place desired in the program. Each label name is followed by a colon (LABEL:) so it is not taken to be a variable. Using GOTO to jump in or out of FOR/NEXT loops or procedures is not possible.

GRAPHMODE [G] — Set graphic mode

```
GRAPHMODE mode
```

For all graphic output, except the PUT command there are four different graphic modes:

mode = 1 (Replace)
: The applicable graphic element (PBOX, LINE, etc) is shown. All that is underneath it, is covered over and replaced. (new point = color mask if the new point AND new point)

mode = 2 (Transparent)
Points are only set where the new graphic element color matches. Where there is no color set, the background remains. The new region appears transparent. (new point = (color of the new point AND new point) OR (color of the old point AND NOT new point)).

mode = 3 XOR (Exclusive OR)
Points are only set where there were previously none. If two graphic elements of the same form are set at the exact same place, the first of the two is erased without disturbing the background. (New point = new point XOR color of the old point)

mode = 4 (Reverse Transparent)
This mode is identical to Mode 2. The difference is that the line ending color is displayed in reverse. If two of the same graphic elements are displayed once in Mode 2 and once in Mode 4, the negative of the first is displayed. (new point = (color of the old point AND new point) OR (color makes of the new point AND NOT new point)).

HARDCOPY [H] — Print monitor screen

HARDCOPY

The current contents of the screen are printed. This command can be broken out of by using the Alternate/Help keys. After about 30 seconds, the interpreter continues working. If the printer is not turned on, or is OFF LINE, this command is automatically terminated after about 30 seconds.

HEX$ — Numeric -> hexadecimal

```
HEX$(expression)
```

Converts a decimal, binary or octal value to the hexadecimal representation returned in a string. `expression` is a variable, a value or numerical expression in one of the other three number systems. The string returned with `HEX$` can be directly output with the `PRINT` command, placed in a string variable or used in a string expression. If a value is given in hexadecimal format, the prefix &H must be used (for example, &HE4FA1B). The interpreter accepts hexadecimal values in this format for numbers represented in another system. Only integers can be represented in this format.

HIDEM [HI] — Turn mouse pointer off

```
HIDEM
```

Effectively disables the mouse pointer. When reading screen information directly into the screen memory with `BLOAD`, the picture is damaged by the mouse pointer mask. This also is made superficially invisible with `DEFMOUSE string$(74,MKI$(0))`.

Although `HIDEM` turns the mouse pointer off, the coordinates can still be read with `MOUSE X,Y,K` or `MOUSEX/MOUSEY`. If a file selector box or alert box is displayed while the mouse pointer is off, the mouse pointer becomes visible for the duration of the box operation. It is removed again afterward. It also reappears automatically when the program ends. This command is the reverse of `SHOWM`.

HIMEM — First byte after the BASIC area

```
HIMEM
```

Returns the first address after the memory area occupied by GfA BASIC (interpreter, program, `FRE(0)` area and variables). This value is needed to get the start address of the unused area when using the `RESERVE` command (block manipulations with `BMOVE`, `BGET`, `BLOAD`).

IF (ELSE) ENDIF [I.E.EN] — Condition test

```
IF condition
...program lines to execute when condition is true...
(ELSE)
...program lines to execute when condition is false...)
ENDIF
```

It is possible to make the execution of program lines dependent on a condition. The commands between the `IF` and `ENDIF` statements are executed if the condition is true. If the `ELSE` option is used, then the commands between the `IF` and `ELSE` are executed when the condition is true, and the commands between `ELSE` and `ENDIF` if the condition is false. `IF` conditions can be nested as deep as desired.

INC — Increment (+1)

INC v_name

Increase the value of a numerical variable or array variable v_name by 1.

INFOW [INF] — Window information line

INFOW handle,string_expression

Determines whether a window info line is opened and sets its contents. A GEM window can contain an information line under the title line. The window identifier handle determines the number of the window to address. The info text string_expression can be passed either as arbitrary text in quotation marks or as a string variable. If you want to use this info line it must be set up with INFOW before the corresponding window is opened with OPENW. If a window is opened without an info line, the INFOW command doesn't affect it until it is closed with CLOSEW and opened again with OPENW. This is the procedure that must be used to remove an info line. string_expression must then be an empty string ("").

INKEY$ Return a single character from the keyboard

`Char$=INKEY$`
(places the pressed character in the variable `Char$`)

`IF/WHILE/UNTIL LEN(INKEY$)`
(condition with keypress, when `INKEY$ > ""`)

`IF INKEY$="Z"`
(condition, when `"Z"` is pressed)

The program is in an endless loop (doesn't stop) until either a ASCII key or another key (arrow, function, etc.) is pressed. In any case only one character at a time can be returned. If no key is pressed, the null string ("") is returned. With ASCII keys, the pressed character is returned. With the other keys a two character string is returned, the first character is always null. The second character contains the pseudo-ASCII code of the pressed key. As the syntax variations show, the returned character is not placed in every case in a string variable. `INKEY$` can also be used directly in a condition statement.

```
Do
 char$=Inkey$
 char.1=Asc(left$(char$))
 char.2=Asc(right$(char$)
 char.3=char.1*256 + char.2
 Print at(10,10); spc(69)
 Print at(10,10)
 Print using "! ### ### #####", char$, char.1,
   char.2, char.3
 'pressed character, ASCII-code, Scan-Code and
 ' Word value of the pressed key
 Repeat
 ' clear the keyboard buffer
 Until Inkey$=""
 Pause 10
Loop
```

INP — Read a byte from a peripheal

```
INP (Port)
INP (#Channel_number)
```

INP waits until a single byte is either sent from a disk file or from a port. Either the data channel number is given or a port number. CAUTION: This command cannot be broken out of with Control/Shift /Alternate. If it is called and a byte is not available, the system must be reset.

The ports have the following identification numbers:

0	=	Printer port	(LST:)
1	=	Auxiliary port	(AUX:)
2	=	Console port/keyboard	(CON:)
3	=	MIDI port	(MID:)
4	=	Keyboard processor	(IKB:)
5	=	Video port/monitor	(VID:)

```
X=INP(2)
```

Waits for a key press and saves the ASCII value in the variable X.

```
PRINT INP(#1)
```

Prints the ASCII value of the byte at the current file pointer location for the file open on channel #1.

INP? Return port input status

```
INP?(port)
```

Indicates whether a byte can be read from the given port (see `INP`). `port` specifies the identifier of the port whose status is returned.

0 =	LST: printer
1 =	AUX: serial interface
2 =	CON: keyboard and screen
3 =	MID: MIDI in

If a byte can be read, the value -1 (true) is returned, else 0 (false).

INPUT [inp] Input of data

```
INPUT v_Name
```
(without text message)

```
INPUT "Text"[;,]v_name
```
(with text message)

```
INPUT v_name1$,v_name2...
```
(several variables)

```
INPUT "Text";v_name1,v_name2...
```
(with text message)

The program stops and waits for the data to be input. This data is then transferred to the appropriate variables.

"Text" is a desired character string (always in quotation marks), which can optionally be used. The text message is then displayed before the input. v_name represents the name of a desired variable which should be input. The character between "Text" and v_name determines the position of the cursor after "Text" has been displayed.

; Question mark and a space after "Text".

, Input begins directly to the right of the text.

The input must be confirmed with the Return key. If the data type doesn't match the variable, the bell sounds, and input must be re-entered.

A comma in the input line separates variables for the same input command. If the input command has several variables, these can be truncated by using commas. In these cases, data can also be truncated by using the Return key. Commas should also be used to assign the proper input values to the variables in the INPUT command.

With string input, the maximum length is 255 characters. The input can be corrected with the Delete, Backspace and arrow keys.

The input of special characters is accomplished in the following manner:

The Alternate key together with another key.

The Control and S key together, followed by another key.

The Control and A key together, followed by the desired ASCII code of the character. Characters with a ASCII value between 0 and 32 must be terminated with the Return key.

INPUT$ Input of a character string

```
A$=INPUT$(count)
```
(Keyboard input)

```
A$=INPUT$(count,#channel)
```
(Input from file)

The program stops and waits for the input of `count` characters from the keyboard or a file. With `count` a maximum string length can be given. When this many characters have been input, the program automatically resumes. Quitting with the Return key is only necessary if the string is shorter than specified in `count`. Editing the input is the same as described in the `INPUT` command. The option parameter `#channel` can be used to obtain input from a file open on channel number `channel`. The maximum length of the string is 32767 characters.

INPUT# [INP] Read data from a data channel

```
INPUT# Channel_number, v_name
INPUT# Channel_number, v_name1$, v_name2...
```

`INPUT#` allows reading data from the given file number. This data is placed in variables as it is read in. `Channel_number` is the file identification number (0-99). `v_name` stands for any desired variable which the data should be read into. Commas separate the variables from one another. The variables must be the same type as the data read. If this is not the case, an error message appears. Strings have a maximum length of 255 characters. For writing data, see the `WRITE#` command.

INSTR — Search for a character string in another string

```
INSTR([start_position,] string, search_string)
```

Returns a value in the position of a given string within another string. `string` is the string or string variable which is searched. `search_string` is the text expression or string variable which is searched for in `string`. If the `start_position` option is not used, the search begins at the first character of `string`, otherwise the search begins at `start_position`. If `search_string` is found in `string`, the position of `search_string` in `string` is returned. If the `search_string` is not found, a value of 0 is returned. If both strings are null (""), a 1 is returned. It is important to remember that a distinction is made between upper and lowercase letters. The value of `INSTR` can be directly output, assigned to a variable or used in a condition statement.

INT — Integer function

```
INT(x)
```

Turns a real number into an integer. The next smallest integer is returned. The argument `x` is any desired positive or negative number. `INT` ignores any value after the decimal point. The result is always an integer.

INTIN — VDI - Integer input block

Start address of a memory block for storing the function specific parameters passed to a VDI function.

If the function must access parameters, they must be passed in the INTIN array with DPOKE as integers (two bytes per element). The number of elements and their meanings depend on the function called.

INTOUT — VDI - Integer output block

Start address of a memory block in which the function specific parameters are returned from a VDI function.

If the function returns parameters, they can be read from the INTOUT array with DPEEK as integers (two bytes per element). The number of elements and their meanings depend on the function called.

KILL [K] — Erase a file

```
KILL "filename"
```

A file is erased from the diskette. filename is the name of the file to erase. Placing the filename in quotes is optional.

LEFT$ — Return left justified characters

```
LEFT$(string[,count])
```

Returns a certain number of characters of a given string beginning with the first character of the string. `string` is a string variable or expression, followed by a comma and then the number of `count` characters to read from the string. `count` can also be a variable. If `count` is not given, the first character of the string is returned. If `count` is larger then the length of `string`, the entire string is returned. In the case when `string` is null (""), a null string is returned. `LEFT$` can be displayed directly, placed in a string variable or used in a string expression.

LEN — Returns the length of a string

```
LEN(v_name$)
```

Determines the length of the given string. `v_name$` is the name of a string variable whose length is determined. `LEN` can be directly output, placed in a variable or used in a condition statement.

LET [LE] — Set variables

```
LET v_name=expression
LET v_name$=text_expression
```

LET sets variable data. expression can be a value or a desired numeric, alphanumeric, boolean or string variable of the same type as the variable to set (v_name or v_name$). The LET command is optional. The variable before the equals sign contains the value of the expression behind the equals sign. The meaning of the LET command is that the variable name v_name is set to a desired value. There are a few reserved variables in GfA BASIC (for example, PI, ERR, FATAL, TIMER, TIME$, DATE$) that cannot be set with the LET command.

LINE [LI] — Display a line

```
LINE X1,Y1,X2,Y2
```

Connects two points with a line. The two coordinate pairs X1/Y1 and X2/Y2 are connected with a straight line. See DEFLINE for setting the line type.

LINE INPUT [LI INPUT] Character string input

```
LINE INPUT"Text"[;,]v_name$
```
(with text)

```
LINE INPUT v_name$
```
(without text)

```
LINE INPUT v_name1$, v_name2$...
```
(several variables)

```
LINE INPUT "Text";v_name1$, v_name2$...
```
(both)

Allows the input of character strings during program execution. With the INPUT command commas are used to separate different variables, commas with the LINE INPUT command become part of the string. If several variables are input, each must be terminated with the Return key. The optional parameter "Text" allows the display of any desired text before the variables are input. Editing is accomplished as described in the INPUT command.

LINE INPUT# [LI INPUT]

Read in a character string

```
LINE INPUT# Channel_number, v_name$
```

LINE INPUT# read in a character string from the given file. While the INPUT# command uses commas to separate variables, commas are part of the text with the LINE INPUT# command. Channel_number identifies the file (0-99) to read from. v_name$ is any desired string variable.

LIST [LIS] Save(ASCII format) or list program

```
LIST
```
(Listing on screen)

```
LIST "Filename"
```
(Listing in a file)

LIST lists out the program text on the monitor or as ASCII text in a disk file. If just the command name is given, the program is displayed on the monitor. This listing can be stopped by using the break function Control/Shift/Alternate. The listing is placed in a ASCII file when the command is given with a filename. This form of the command is absolutely identical to the editor function SAVE,A. A program saved in this manner can be merged with another using the editor function MERGE. Placing the filename in quotes is optional. If no extension is given, the interpreter supplies .LST.

LLIST [LL] Print a program listing

```
LLIST
```

The program currently in the work memory is printed out. This command can only be broken out of by turning the printer off or placing it OFF LINE. After about 30 seconds the interpreter is ready again. If the printer is not turned on, or is OFF LINE, this command is automatically terminated after about 30 seconds.

LOAD [LOA] — Load a program

```
LOAD "Program_name"
```

Loads a desired BASIC program into the work memory. `Program_name` is the name of the program to load. If the program name doesn't have an extension, the interpreter uses the extension of .BAS. This command is basically identical to the `LOAD` function in the Editor menu.

LOC — File pointer position

```
LOC (#Channel_number)
```

The distance between the beginning of the file and current position of the file pointer is returned. Every open file has a read/write pointer which points to the current disk location. This command returns the value of that pointer, which is always relative to the beginning of the file. The file pointer is always pointing to the byte returned by `LOC`. `Channel_number` is the file identifier which should be checked.

LOCAL [LOC] — Declare local variables

```
LOCAL lv_name
```
(local variable)

```
LOCAL lv, lv%, lv$...
```
(local variable list)

Inside of procedure, local variables can be declared. `LOCAL` variables have the characteristic that can be used exclusively within the procedure that they were declared in. If a variable with the same name is used outside of this procedure, the interpreter distinguishes between the two. The interpreter uses a `LOCAL` variable only within the procedure where it is declared. A list of variables of different types can also be given with this command.

LOF — Determine length of a file

```
LOF (#Channel_number)
```

`LOF` returns the length of a file in bytes. `Channel_number` is the number of the file to check.

LOG, LOG10 — Logarithm functions

`LOG(x)`
(natural logarithm)

`LOG10(x)`
(common logarithm)

Calculates the natural or common logarithm of a number. The logarithm of a number `x` is the exponent to a base (here `E` or 10). This power calculation then gives the number `x`. The constant `E` is equal to 2.7188182... The numeric expression `x` must always be larger than 0!

LPOS — Determine print head location

`LPOS(Dummy)`

When printing a certain number of bytes (depending on the printer type) are collected in a buffer before being printed. This command returns the virtual printer head position in this buffer. It is not necessarily the same as the actual printer head position.

LPRINT [LPR] Print data on printer

```
LPRINT
LPRINT "Text"
LPRINT "Text"[;,']"Text"[;,']V_name[;,'] V_name$[;,']...
```

LPRINT prints data and/or text and/or blank lines (or spaces) on the printer. It is used in the same fashion as PRINT. However, it is not possible to position the print head by using AT. Using various escape sequences (CHR$(27) -- see printer manual), various control commands can be sent to the printer.

LSET [LS] Set a left justified string

```
LSET string_var$="string expression"
```

Places a given left justified string in a string variable. string expression can be any desired character string or string variable. The contents of this string are then placed left justified in string_var$. The length of string_var$ is never changed. If the length of string_var$ is smaller then the length of string_expression, the string_expression is truncated to the length of string_var$. If string_var$ is longer then the string_expression, the rest of string_var$ is filled with blank spaces.

MAX — Returns the largest value in a list

`MAX(expression1,expression2,expression3,...)`

Returns the largest value from the expressions or using the ">" operator, the largest string. `expression` can be any desired numeric or text expression, string or variable. All expressions must be of the same type. With numeric comparisons, the largest number is returned. In string comparisons, the largest string is returned after all initial characters have been compared. If two characters have the same ASCII value, the next two characters are checked until the characters do not match or there are no more characters in one of the strings. In the first case, the result is the largest of the last checked characters. In the second case, the largest is the longest string.

MENU(index) — Event buffer

`Menu(index)`

This function is used to obtain the relevant data for an action depending on the event which has occurred. Behind this function is a one-dimensional numerical array that the interpreter stores various data for each occurring event. Each element of this array can be read by specifying the appropriate `index`. The meanings of the entries in points 1-8 can vary from case to case.

The contents of `MENU(index)`

`Menu(-1)` = Address of the current menu object tree
`Menu(0)` = Index of the selected menu option

`Menu(1)` = Event number

10= (WM_SELECTED) drop-down menu is selected
Menu(4) = title index
Menu(5) = menu option index

20= (WM_REDRAW) redraw window area
Menu(4) = handle
Menu(5) = X-coordinate
Menu(6) = Y-coordinate
Menu(7) = width
Menu(8) = height

21= (WM_TOPPED) a window should be activated
Menu(4) = handle

22= (WM_CLOSED) the close field is selected
Menu(4) = handle

23= (WM_FULLED) full field is selected
Menu(4) = handle

24= (WM_ARROWED) window contents should be scrolled
One of the four arrows or one of the two scroll bars is selected.
Menu(4) = handle
Menu(5) = index of the selected object:
0 = whole page up
1 = whole page down
2 = one line up
3 = one line down
4 = whole page left
5 = whole page right
6 = one character left
7 = one character right

25= (WM_HSLID) horizontal slider is moved.
Menu(4) = handle
Menu(5) = slider position relative to surrounding box (0-1000)

26= (WM_VSLID) vertical slider is moved
Menu(4) = handle
Menu(5) = slider position relative to surrounding box (0-1000)

27= (WM_SIZED) size field is selected
Menu(4) = handle
Menu(5) = old window X-coordinate
Menu(6) = old window Y-coordinate
Menu(7) = new window width
Menu(8) = new window height

28= (WM_MOVED) grey move bar is selected
Menu(4) = handle
Menu(5) = new window X-coordinate
Menu(6) = new window Y-coordinate
Menu(7) = old window width
Menu(8) = old window height

Menu(4) = handle or menu bar index
Menu(5) =X-coordinate or slider position
Menu(6) = Y-coordinate
Menu(7) = width
Menu(8) = height
Menu(9) = event flag

32= Mouse button event

34= No event

35= Keyboard event

48= Arrow or scroll bar event

50= Menu corner point, move bar or slider event
Menu(10) = mouse X-coordinate at time of event
(relative to upper let screen corner)

`Menu(11) =` mouse Y-coordinate at time of event
(relative to upper let screen corner)

`Menu(12) =` mouse button status
(1=left;2=right;3=both)

`Menu(13) =` "Switch" key status
Right shift key = 1
Left shift key = 2
Control key = 4
Alternate key = 8
Combinations of all four keys are possible
e.g. "Control" + left "Shift" = 6

`Menu(14)` = Key code (16-bit value)
High = Scan code
Low = ASCII code

`Menu(15)` = Number of mouse clicks at the time of the event

MENU

Set menu option attributes

`MENU menu,value`

This command allows you to change the attributes of a menu (active, checked). `menu` specifies the index of the menu option whose attribute is set (see `MENU(index)`). By specifying the appropriate `value`, the given menu option can be activated, deactivated, or a mark can be placed in front of it or removed from it.

Value = 0 -> delete "check mark"
Value = 1 -> set "check mark"
Value = 2 -> deactivate menu option
Value = 3 -> activate menu option

If you want to be able to put a check mark in front of a menu option, two spaces must be placed before the name of the option when it is set with the MENU command `menu_text$`.

MENU — Create menu line

```
MENU string_array()
```

A string array is defined which contains the desired text for the menu titles and their menu options. `string_array()` is a one-dimensional string array which must have at least as many elements as menu titles and options are defined, plus an additional 20 elements for storing various organization strings.

The first menu has the following construction:

`String 1`
: First menu title.

`String 2`
: Any string. A program function can be assigned to this menu option. Since it is the only usable menu option in this menu, it is best suited for displaying program information.

`String 3`
: Row of dashes. The number of dashes determines the width of the menu. Since this menu is used to access the desk accessories, the maximum accessory title length should be taken into account here.

`String 4 to String 9`
: Six blank strings to serve as place holders for the accessories. No null strings ("") can be passed.

`String 10`
: Null string ("") to mark the end.

The remaining, user definable menus are appended to this menu structure. These are constructed such that the individual menu option designations follow the menu titles. A null string ("") must be used to mark the end of each menu. Two null strings must be appended to the end of the array. If a menu option is preceded by a dash, it appears as disabled in the menu. If check marks are used, two spaces must be inserted in front of the appropriate menu option names.

MENU KILL

Remove the menu line

The drop-down menus are deactivated. After using this command, menu selections cannot be made. The menu line text is not deleted. To reactivate the menus, use the ON MENU command again. The menu text array cannot be changed in the meantime.

MENU OFF

Invert menu title

MENU OFF

Displays an activate menu title in normal mode (black on white) again. MENU OFF should always be used when a menu is opened and a menu option is selected. If no option is selected and a mouse button is pressed outside the menu, this measure is not necessary.

MID$ Return a string from the middle of another

```
MID$(string, start[,count])
```

Returns a set number of characters (from a desired starting position) from a character string. `string` can either be text or a string variable. The `start` parameter sets the location of the copy. (For example `start=5` --> start copying at the 5th character). `count` is the number of characters to copy from the string. Both `start` and `count` can be variables. If `count` is not given, all characters from `string` are copied starting at the `start` position. When `count` is larger then (`LEN(string) - start`), all characters from `start` onward are returned. If `string` is null, a null string is returned. `MID$` can be directly output, placed in string variables or used as part of a string expression.

MID$() Replace substring

```
MID$(var$,start,number) = string_expression
```

With this command, part of an existing string can be replaced by a string expression. `var$` is the name of the string variable whose contents are replaced, `start` contains the location in the old string at which the new string is placed, and `number` is an optional parameter which indicates the maximum number of characters to be replaced. If `number` is not used, the number of characters replaced are either as many as are contained in `string_expression`. If `string_expression` is longer than the number of characters remaining in the string from `start` on, then the rest of the characters in `var$` are replaced. The original length of the string is not changed by this operation.

MIN — Returns the smallest value in a list

```
MIN(expression1,expression2,expression3,...)
```

Returns the smallest value from the expressions or using the "<" operator, the smallest string. `expression` can be any desired numeric or text expression, string or variable. All expressions must be of the same type. With numeric comparisons, the smallest number is returned. In string comparisons, the smallest string is returned after all initial characters have been compared. If two characters have the same ASCII value, the next two characters are checked until the characters do not match or there are no more characters in one of the strings. In the first case, the result is the smallest of the last checked characters. In the second case, the smallest is the shortest string.

MKDIR [MK] — Create a directory

```
MKDIR "Directory name"
```

`MKDIR` creates a new directory in the current directory. `Directory name` is the name of the new directory and can also be a pathname if the directory should be created some place other than the current directory.

MKI$, MKL$, MKS$, MKF$, MKD$

Format strings

```
MKI$(16 bit_integer_number)
MKL$(32 bit_integer_number)
MKS$(real_number)
MKF$(real_number)
MKD$(real_number)
```

A number is converted into a desired string format. The number is converted into the appropriate string form based on its size and type.

MKI$	converts a 16 bit integer value into a 2 character string
MKL$	converts a 32 bit integer value into a 4 character string
MKS$	converts a real number in Atari BASIC format into a 4 character string
MKF$	converts a real number in GfA BASIC format into a 6 character string
MKD$	converts a real number in MBASIC format into a 8 character string

The desired value can be a numeric expression or a variable. The returned string can be directly output, placed in a variable or used in a condition statement. These functions are inverses of CVI, CVL, CVS, CVF and CVD.

MONITOR — Call machine language monitor

```
MONITOR(parameter)
```

If GfA BASIC is started from a resident machine language monitor, this command can be used to call the monitor. `parameter` can be used to pass an optional value to the monitor. Only monitors or debuggers which fulfill certain criteria can be used. One of these is "SID", from the Digital Research development package.

MOUSE [M] — Return the (entire) mouse status

```
MOUSE Xpos, Ypos, Buttons
```

Reads the position of the mouse and the state of the two mouse buttons. The X- and Y-coordinates of the mouse pointer are returned in the variables `Xpos` and `Ypos`. The state of the mouse buttons is returned in the `Buttons` variable.

0 = no buttons pressed
1 = left button pressed
2 = right button pressed
3 = both buttons pressed

MOUSEX/Y/K — Mouse status

MOUSEX
(X position)

MOUSEY
(Y position)

MOUSEK
(Mouse button status)

Reads the position of the mouse and the state of the two mouse buttons individually. The functions read the desired status component individually. The returned values can be stored in a variable, used in a condition or printed directly.

MUL [MU] — Multiplication command

```
MUL v_name1, v_name2
MUL v_name1, const
```

Multiplication of variables with result assignment. `v_name1` and `v_name2` can be numeric variables or array variables. `const` is a numeric constant. The value before the comma is multiplied by the value after the comma and the result is placed in the variable `v_name1`.

NAME [NA] — Change the name of a file

```
NAME "Old_name" AS "New_name"
```

The name of a current file can be given a new name. Both names can either be string constants, string variables or a combination of the two. This is a command in which a data name is expected. The drive specification must be the same for both names. If the command is executed in the current directory, the drive need not be specified. In a given directory, the name of a file in another directory can be changed by giving the complete pathname of the file and the new name of the file (also a complete pathname in the same directory).

NEW — Erase program memory

```
NEW
```

The NEW command erases every portion of the program in the BASIC work memory. The program, its variables and all initialization is erased. The memory can now be used for another application.

OCT$ — Numeric -> octal

```
OCT$(expression)
```

Converts a decimal, hexadecimal or binary value to the octal representation returned in a string. `expression` is a variable, a value or numerical expression in one of the other three number systems. The string returned with `OCT$` can be directly output with the `PRINT` command, placed in a string variable or used in a string expression. If a value is given in octal format, the prefix &O must be used (for example, &O25526277). Only integers can be represented in this format.

ODD — Test for an odd number

```
ODD(number)
```

The argument is tested to see if it is odd. The value -1 (true) is returned if the value is odd. If it is not, the value 0 (false) is returned.

ON BREAK — Break handling

ON BREAK GOSUB procedure_name
ON BREAK CONT

If during program execution the break key combination Control/Shift/Alternate is pressed, this branches to procedure_name. The command can be used any place in a program as often as desired. It is also possible to activate this according to the specific situation, or to completely disable the break key by using the CONT parameter.

ON ERROR GOSUB — Error handling

ON ERROR GOSUB procedure_name
(branch on error)

ON ERROR
(output of interpreter message)

Branches to the given procedure as soon as an error is encountered, or turns on the output of interpreter error messages. procedure_name is the name of the desired procedure to branch to when an error occurs. ON ERROR GOSUB must be executed before the error occurs. After an error handling procedure is executed, the interpreter reactivates its normal error handling mode. If no error is encountered, but normal mode should be reactivated, the ON ERROR command is used. The interpreter displays an error window containing the error message, exits the routine and breaks out of the program.

ON...GOSUB — Branch to procedures

`ON value GOSUB proced.1, proced.2, proced.3,...`

ON GOSUB decides which procedure it should branch to, based on a given value. The value starts at 1 and can go in single increments as high as desired. For example:

```
VALUE  >= 1 < 2  ---> Procedure1
VALUE  >= 2 < 3  ---> Procedure2
VALUE  >= 3 < 4  ---> Procedure3
...etc.
```

If a value is encountered which is larger than the number of procedures, or the value is exactly 0, the program line following ON...GOSUB is executed.

ON MENU — Branch to event handler

ON MENU

Enables event trapping. This command is used when an event (mouse movement, keyboard, window actions, etc.) has occurred. This should occur in program locations which are repeated often (inside program loops) because the message buffer is continually updated. If the program doesn't react quickly enough to the event and read it from the buffer, newer buffer entries can overwrite the entry and yield an incorrect result. If the command is not used, none of the procedures defined with one of the ON MENU GOSUB commands is called.

ON MENU BUTTON GOSUB — Mouse event

ON MENU BUTTON numb.,button,status GOSUB proced.

Branch to a procedure if a predefined mouse button is pressed multiple times (double or triple click). number specifies the maximum number of clicks which are regarded. This affects only the number of clicks which are recorded. button indicates which button event is monitored:

0 = no button
1 = left mouse button
2 = right mouse button
3 = both mouse buttons simultaneously

status defines the status of the selected button which leads to the event:

0 = no button pressed
1 = left button pressed
2 = right button pressed
3 = both buttons pressed simultaneously

Since the correctness of the information above cannot be verified and the subsequent analysis was rather complicated, we have listed various ways of using this command which should cover the variations you encounter. Only a maximum of two clicks is regarded here since a triple click is quite difficult for most people to accomplish.

ON MENU BUTTON 2,0,1 GOSUB Button
The procedure is executed for all button combinations.

ON MENU BUTTON 2,1,1 GOSUB Button
The procedure is executed only when both buttons or the left button is pressed.

`ON MENU BUTTON 2,2,1 GOSUB Button`
The procedure is executed only when the left button is pressed.

`ON MENU BUTTON 2,3,1 GOSUB Button`
The procedure is executed only when the left button is pressed.

`ON MENU BUTTON 2,2,3 GOSUB Button`
The procedure is executed only when the right button or both buttons are pressed simultaneously.

`ON MENU BUTTON 2,3,2 GOSUB Button`
The procedure is executed only when the right button is pressed.

`ON MENU BUTTON 2,3,3 GOSUB Button`
The procedure is executed only when both buttons are pressed simultaneously.

The conditions associated with the event are placed in the procedure. The number of actual mouse clicks can be read from `Menu(15)`. The X/Y-coordinates of the mouse at the time of the event can be read from `Menu(10)` for X and `Menu(11)` for Y.

ON MENU GOSUB — Branch on menu selection

`ON MENU GOSUB procedure_name`

The procedure defined with this command is executed whenever one of the active drop-down menu options is selected. The `MENU` function can then be used to determine which option is selected and the procedure can react as appropriate.

ON MENU I/OBOX GOSUB — Mouse event

```
ON MENU IBOX Id,Xp,Yp,W,H GOSUB procedure_name
ON MENU OBOX Id,Xp,Yp,W,H GOSUB procedure_name
```

These commands define a procedure which is called when the mouse pointer enters or leaves one of the four possible screen areas. Each of these commands can define two screen areas which are watched for mouse events causing the mouse to move in (IBOX) or out of (OBOX) the area, assuming that event trapping is enabled with ON MENU. If such an area is defined with ON MENU OBOX GOSUB, the specified procedure is called when the mouse pointer moves out of this area. With ON MENU IBOX GOSUB, the call takes place when the mouse pointer enters the specified area. `procedure_name` stands for the procedure which is called. `Id` is an identifier and determines which of the two independent fields are watched. The X/Y-coordinate pair describing the upper left corner of the area are given in `Xp` and `Yp`. The width and height of this field are then defined with `W` and `H`, respectively.

ON MENU KEY GOSUB — Keyboard event

```
ON MENU KEY GOSUB procedure_name
```

This command defines a procedure which is executed when a keyboard event occurs. If ON MENU is used to enable menu and window event trapping and if a procedure is defined with ON MENU KEY GOSUB, the event buffer can be read in this procedure (see MENU). The scan code and ASCII codes of the key or combination of keys pressed by the user are in `Menu(14)`.

ON MENU MESSAGE GOSUB — Multi-event

ON MENU MESSAGE GOSUB procedure_name

This command defines a procedure which is called when an event occurs in the message buffer. If ON MENU is used to enable menu and window event trapping and if a procedure is defined with ON MENU MESSAGE GOSUB, the event buffer can be read in this procedure (see MENU) and the procedure can react appropriately.

OPEN [O] — Open a data channel

OPEN "Mode", #Channel_number, "Filename"

A data channel (disk files, virtual files) can be opened for reading, writing, appending, updating or random access.

Mode:

O (Output)	Opens a file for writing, or if the given file is not available, initializes the given channel number.
I (Input)	Opens an existing file for reading.
A (Append)	Opens an existing file and places the file pointer after the last byte in the file. All data written to this file are placed at the end of the previous data.
U (Update)	In this mode a file can be both written to and read from.

R (Random) A Random Access File is created. With random access files it is possible to read or write records at any location in the file. This is done by using the GET# and PUT# commands. After the file is created, the record must be defined using the FIELD# command. The OPEN command also must be used with an additional parameter:

```
OPEN "R", #Channel_number, "Filename", r_length
```

Channel_number
This is the identification number of the file to open (0-99). This is then used with other commands such as PRINT# or INPUT# to access the file.

Filename A simple filename, a pathname or a virtual file (port).

CON: =	Console port
LST: =	Printer port
PRN: =	Printer port
AUX: =	Auxiliary port
MID: =	Midi ports
VID: =	Monitor (Video)
IKB: =	Keyboard processor

The complete word before the equal sign is used as the filename. MODE is not used with virtual files.

OPENW [O W] — Open window

OPENW handle,X_pos,Y_pos
(set intersection)

OPENW handle
(open/update)

Opens a GEM window or causes it to be displayed as the current window. The handle identifier determines the window addressed. The numbers 1, 2, 3 and 4 stand for each of the four possible windows. The windows are opened next to each other without overlapping:

Window 1 Window 2

Window 3 Window 4

The windows touch each other at the screen point defined by the coordinate pair X_pos/Y_pos. This intersection need be defined only once and can be changed when opening or updating a window by specifying new values for X_pos/Y_pos. This moves the contact axes of the windows to the new intersection. If a window is already open and is not current, it can be made the current window with OPENW.

When opening a new window, an invisible window is automatically opened with handle = 0. At the end of the program or if the entire TOS output screen is activated, this window should be closed first with CLOSEW 0. If no additional GEM windows are open, the origin for graphic output can be changed with the OPENW 0,X_pos/Y_pos command.

```
Openw 0
Deffill ,2,4
Pbox 0,0,639,380
For I=1 To 4
 Dpoke Windtab+2+(I-1)*12,&X111111111111
 Titlew I," TITLE line "+Str$(I)
 Infow I," INFO line "+Str$(I)
 Openw I
 Clearw I
Next I
Count=4
On Menu Message Gosub M.essage
Do
 A$=Inkey$
 If A$=>"1" And A$=<"4"
  Openw Val(A$)
  Clearw Val(A$)
  Inc Count
 Endif
 On Menu
 Mouse X,Y,K
 Xr=Dpeek(Windtab+8+(Menu(4)-1)*12)-20
 Yr=Dpeek(Windtab+10+(Menu(4)-1)*12)-60
 If K=1 And X=>0 And Y=>0 And X<Xr And Y<Yr
  Print X''Y
 Endif
Loop
Procedure M.essage
 If Menu(1)=20
  Print "Area ";Menu(4);": ";Menu(5)'Menu(6)'Menu(7)'Menu(8)
 Endif
 If Menu(1)=21
  Infow Menu(4),"(TOPPER) Window "+Str$(Menu(4))+
   "selected"
  Openw Menu(4)
  Clearw Menu(4)
```

```
Endif
If Menu(1)=22
 Closew Menu(4)
 Dec Count
 If Count=0
  Alert 2,"Exit program ?",1,"OKAY|NO",D%
  If D%=1
   Closew 0
   Cls
   Edit
  Else
   Alert 1,"Open window:|Keys 1 - 4",1,"OKAY",D%
  Endif
 Endif
Endif
If Menu(1)=23
 Closew Menu(4)
 Dpoke Windtab+4+(Menu(4)-1)*12,0
 Dpoke Windtab+6+(Menu(4)-1)*12,19
 Dpoke Windtab+8+(Menu(4)-1)*12,639
 Dpoke Windtab+12+(Menu(4)-1)*12,380
 Infow Menu(4),"(FULLER) Full size !"
 Openw Menu(4)
 Clearw Menu(4)
Endif
If Menu(1)=24
 Infow Menu(4),"(ARROWS) Arrow:"+Str$(Menu(5))
Endif
If Menu(1)=25
 Infow Menu(4),"(SLIDE)H-slider : "+Str$(Menu(5))
Endif
If Menu(1)=26
 Infow Menu(4),"(SLIDE)V-slider : "+Str$(Menu(5))
Endif
If Menu(1)=27
```

```
  Dpoke Windtab+8+(Menu(4)-1)*12,Menu(7)
  Dpoke Windtab+10+(Menu(4)-1)*12,Menu(8)
  Closew Menu(4)
  Infow Menu(4),"(SIZER) Window size changed"
  Openw Menu(4)
  Clearw Menu(4)
 Endif
 If Menu(1)=28
  Dpoke Windtab+4+(Menu(4)-1)*12,Menu(5)
  Dpoke Windtab+6+(Menu(4)-1)*12,Menu(6)
  Dpoke Windtab+8+(Menu(4)-1)*12,Menu(7)
  Dpoke Windtab+10+(Menu(4)-1)*12,Menu(8)
  Closew Menu(4)
  Infow Menu(4),"(MOVER) Window was moved"
  Openw Menu(4)
  Clearw Menu(4)
 Endif
Return
```

OPTION — Set compiler options

```
OPTION "command"
```

This command can be used to set options on the GfA BASIC compiler. Please see your compiler documentation.

OPTION BASE — Set array start index

OPTION BASE 0
(start index = 0)

OPTION BASE 1
(start index = 1)

The start index of arrays in the program can be set to either 0 or 1. Since it is sometimes somewhat complicated or undesirable to work with zero-elements in arrays, the starting index can be set to 1 with this command, or back to zero again. This definition applies to all arrays in the program which are defined at the time this command is used. The start index can be changed multiple times in a program without having to worry about losing data.

If a program starts out with `OPTION BASE 1`, the arrays are filled, and the programmer adds the zero element with `OPTION BASE 0` then the entire contents of the arrays are shifted one element down so that element 1 becomes element 0, and so on. In the reverse case (starting with zero elements), all elements are shifted up one when the change is made to one-based arrays.

OUT [OU] — Write a single byte to a peripheal

```
OUT Port, byte
OUT #Channel_number, byte
```

A single byte value can be sent to a given data channel or periphal port. `Port` is the number of the desired periphal port (see `INP`). Caution: Sending an invalid opcode to the IKB: (Intelligent Keyboard) causes the computer to crash. `Channel_number` is

the file identifier (0-99) of the desired file. `byte` is either sent to the port or written to the file at the current file pointer position.

`OUT 5,10` Sends the LF character (ASCII value 10 -- control character LF) to the current monitor cursor position. (The control characters CHR$(0) to CHR$(31) can also be used to position the cursor).

`OUT #1, 13` Writes a Carriage Return at the current file pointer position to the file opened on channel #1.

OUT? Determine port output status

`OUT? (port)`

Returns information about whether the specified port (see `OUT`) is ready for output. `port` is the identifier of the port whose status is returned.

0 = LST: printer
1 = AUX: serial interface
2 = CON: keyboard and screen (console)
3 = MID: MIDI in

If a byte can be outputted, the value -1 (true) is returned, else 0 (false).

PAUSE [PA] — Wait function

```
PAUSE duration
```

PAUSE waits for a certain amount of time. The parameter duration sets how long the program waits. This is in 1/50 second (1 second pause = PAUSE 50).

PEEK/DPEEK/LPEEK — Read memory contents

```
PEEK(address)
```
(reads one byte)

```
DPEEK(complete_address)
```
(reads two bytes (word))

```
LPEEK(complete_address)
```
(read four bytes (long))

Reads in the given format contents of the given memory address. It is important to notice that DPEEK and LPEEK must be given an entire address, otherwise an error message appears. address can be given as a value, a numeric expression or a variable. The returned value can be directly output, placed in a variable or used in a condition statement. These functions are inverses of POKE, DPOKE and LPOKE or SPOKE, SDPOKE and SLPOKE.

PI — Circle number

```
PI
```

Returns the value of the number PI. The number is a geometric constant and is the result of dividing the circumference of a circle by its diameter. PI=3.141592654...

The value PI is necessary when working with circular geometric forms, or calculating areas or volumes of circular objects.

PLOT [PL] — Display points

```
PLOT x_pos, y_pos
```

Displays a graphic point. The coordinate pair x_pos/y_pos sets the place on the screen where the point should be set. PLOT is not influenced by the line width definition set with DEFLINE, with the exception that line beginnings and endings are defined as round.

POINT — Return color value for a screen point

`POINT x_pos, y_pos`

Checks a screen point for its color value. `x_pos/y_pos` is the coordinate pair that should be checked. The following values are valid for this function:

low resolution	0 to 15
medium resolution	0, 1, 2 or 3
high resolution	0 or 1

This value is the same as the parameter for `COLOR`. The returned value can be directly output, placed in a variable or used in a condition statement.

POKE/DPOKE/LPOKE [PO, DP, LP] — Memory command

`POKE address, byte`
(writes one byte)

`DPOKE complete_address, word`
(writes two bytes)

`LPOKE complete_address, long`
(writes four bytes)

Places the given value, in the given memory location with the given format in user mode. `POKE` places a single integer byte and can be in the range of -256 to 255.

DPOKE places two bytes in memory, one after the other. The range for the word argument is -65536 to 65535. LPOKE places four bytes in memory, one after the other. The range for the long argument is -2147483648 to 2147483647. It is important to notice that DPOKE and LPOKE must be given complete addresses, otherwise an error message appears. Both parameters can be values, numeric expressions or variables. These commands are inverses of PEEK, DPEEK and LPEEK in user mode.

POLYFILL [POLYF] Display filled in polygon

```
POLYFILL pts,xp(),yp()
POLYFILL pts,xp(),yp(),OFFSET xdiff,ydiff
```

A desired figure is displayed, in which the closed in areas are filled in. This command is executed in the same fashion as POLYLINE. The only difference is that closed in areas are filled with the current fill in pattern. The area between overlapping patterns is not filled in with this command.

POLYLINE [POL] Display a polygon

```
POLYLINE pts,xp(),yp()
POLYLINE pts,xp(),yp(),OFFSET xdiff,ydiff
```

Allows any desired line pattern to be displayed. The pts parameter sets the number of points to be connected (max of 128). The x_p() and y_p() fields contain the x and y coordinates of the corner points in the figure (x_p(0) and y_p(0) is the first point in the diagram). If a closed figure is desired, the last

coordinate must be the same as the first, namely the first one in the list. (Beginning point = ending point). So then the value of pts for a hexagon would be 7.

Using the OFFSET optional parameter, it is possible to place the figure in a different location than specified in the fields. The output is moved (XDIFF, YDIFF) in the appropriate direction.

To fill in a polygon figure and to mark the corner points of lines, see POLYFILL and POLYMARK.

POLYMARK [POLYM] Polygon corner points

```
POLYMARK pts, xp(), yp()
POLYMARK pts, xp(), yp(), OFFSET xdiff, ydiff
```

The current mark set with DEFMARK is placed at the points given in the POLYLINE array definition. This command is used in the same fashion as POLYLINE except that instead of lines, only the current mark symbol at the defined corners is displayed.

POS Return the current position

```
POS (Dummy)
```

Returns the current row that the cursor is in. Since the output lines can be up to 256 characters long, POS returns a value between 0 and 255. This position must not be taken as the actual cursor position. Dummy can be any desired value. It means absolutely nothing, but still must be given.

PRINT [? or P] Output data

```
PRINT
PRINT "Text"
PRINT,"TEXT"[;,']"Text"[;,']V_name[;,'] V_name$ [;,']...
PRINT AT(R,C) [;,']"Text"[;,']output[;,'] V_name
```

The above syntax diagrams show several different options for using this command. PRINT without text, variables or format characters (i.e. just the PRINT command), outputs a blank line terminated with Carriage Return and Line Feed. The cursor jumps to the start of the next line. When PRINT is followed by a number, variables or a text line; the value, variable contents or text line is displayed, followed by CR and LF. With format characters, the output can be modified as follows:

; Suppresses the printing of the CR and LF following output. The next output is at the very next character position. If the line is longer than 80 characters, the output wraps around to the next line.

, Also suppresses the CR and LF. Output continues at the next Tab position (1, 17, 33, 49, 65).

' The apostrophe is exactly the same as the semicolon, except that a blank space is displayed at the apostrophe location.

With the AT(R, C) option, the cursor can be placed at any desired screen location. The start index is 1 for both dimensions—the upper left corner is AT(1,1). (AT(0,0) is the same as AT(80,25)). In a GEM window the output mode can be set with DEFTEXT and GRAHMODE.

PRINT# [? or P] — Write data to a channel

```
PRINT #Channel_number
PRINT #Channel_number, "Text"
PRINT #Channel_number,
"Text"[;']"Text"[;']V_name[;'] V_name$[;']...
```

This command writes data and/or text and/or blank lines or spaces to an open file. It functions exactly like the PRINT command except that the data is written to the data channel specified by Channel_number (0-99). See also PRINT.

PRINT#, USING — Write formatted data to a channel

```
PRINT #Channel_number, USING "format", list
```

Strings and numbers can be printed in the given format on a open file. The command functions exactly like PRINT USING except that the data is written to the open data channel specified by Channel_number (0-99). For a description of formatting characters, see PRINT USING.

PRINT USING [P USING] Formatted output

```
PRINT USING "format", output$, v_name$, output, v_name
```

This command is given a string which describes how the output should be formatted. The values are prepared as specified and then output. Instead of a text output, string variables can also be formatted and output.

The following format symbols are available:

#	Space for a digit.
.	Shows the position of the decimal point in a number.
+	A plus sign is displayed with the output.
-	If a value is negative, the minus sign appears at this location.
*	The character before the decimal point indicates that all spaces not occupied by digits should be displayed as an "*". After the decimal point, it indicates the number of places a value should be rounded to (< .5 = rounded down, > .5 rounded up).
$	Prefix $ sign.
,	If commas are desired in large numbers, they are indicated using the comma.
^^^	A number is displayed in exponential notation.
!	The first character of the input string is output.
&	The entire input string is output.

\...\ Characters are output from the beginning of the string. The dots indicate the total number of characters to output. (backslashes should be included in the format string).

_ (underline) displays the next character of a string.

PROCEDURE [PRO] Procedure beginning

```
PROCEDURE name
PROCEDURE name (lv_name%, lv_name, lv_name$...)
```

PROCEDURE defines the beginning of a procedure and if desired a list of valid local variables. name can be any desired name, by which the procedure is known. Any and all characters can be used in the PROCEDURE name. In contrast with variables, a procedure name can begin with a digit. If the procedure is declared with a variable list, the GOSUB command must also have a list of data which is enclosed in parentheses. In this list, the variables must be ordered so that the types are the same in both lists. The variables declared in the parameter list are local variables (see LOCAL). That means that these values or strings can be set or changed inside of this procedure. If values are created using local variables, they must be set to global variables within the procedure so that these variables are available outside of the procedure. GfA BASIC also allows procedures to be called recursively. Every procedure ends with the RETURN command.

PSAVE [PS] Save program (list protected)

```
PSAVE "Program name"
```

Saves the work memory under the given name in a coded and list protected form on the diskette. This command is executed in the same fashion as SAVE. The difference is that the program is started as soon as it is loaded and can no longer be listed.

PTSIN VDI - Point input block

Start address of a memory block for storing the function specific point coordinates passed to a VDI function.

If the function requires coordinates, they must be passed in the INTIN array with DPOKE as integers (two bytes per element). The number of elements and their meanings depend on the function called.

PTSOUT — VDI - Point output block

Start address of a memory block in which the function specific point coordinates are returned from a VDI function.

If the function returns coordinates, they can be read from the INTOUT array with DPEEK as integers (two bytes per element). The number of elements and their meanings depend on the function called.

PUT — Display screen portion

```
PUT x,y,v_name$,mode
PUT x,y,v_name$
```

PUT places a screen portion saved in a string variable with GET back on the screen at any desired location as specified with the coordinates (x,y). The size remains unchanged. If the mode option is used, the graphic mode can be set. When this option is not set, the portion is displayed in replace mode.

Mode	Display
0	Erase old background
1	New picture AND background
2	New picture AND NOT background
3	New picture (replaces the background, graphmode 1)
4	(NOT new picture) AND background (graphmode 4)
5	Background (nothing appears)
6	New picture XOR background (graphmode 3)
7	New picture OR background (graphmode 2)
8	NOT (new picture OR background)
9	NOT (new picture XOR background) (reverse XOR)
10	NOT background (reverse background)
11	New picture OR (NOT background)
12	NOT new picture (reverse transparent)
13	(NOT new picture) OR background
14	NOT (new picture AND background)
15	1 (full color)

PUT# [PU]

Write a record

PUT# Channel_channel
(Writes the next record)

PUT# Channel_number, record_number
(Writes the given record)

PUT# writes a record to open "R" files. Channel_number specifies which "R" mode file (0-99) should be written to (see OPEN). With "R" mode files, each record is given a number. Using the record_number option, any desired record in the file can be written. Placing the text in the appropriate FIELD variable can be done with the LSET and RSET commands. The record

number must be chosen so that it is not larger then the number of records currently in the file. The maximum number of records for each file is 65535. If the parameter `record_number` is not used, the next record is written.

QUIT [Q] — Leave the interpreter

```
QUIT
```

`QUIT` is identical to `SYSTEM` which means that the program ends. The interpreter is exited and control is returned to the desktop. There is no verification question before this happens.

RANDOM — Integer random number

```
RANDOM(n)
```

Returns an integer random number within a desired range. The upper limit of the range is given with `n` and can be any desired number. An integer from the range 0(inclusive) to `n`(exclusive) is returned. `n` can also be negative.

RBOX, PRBOX [RB, PRB] Construct a rectangle

RBOX x_left, y_top, x_right, y_bottom
(empty rectangle)

PRBOX x_left, y_top, x_right, y_bottom
(filled in rectangle)

Displays a rectangle with rounded corners either empty or filled in with a pattern. The command is given two coordinate pairs (x_left/y_top and x_right/y_bottom). They are the diagonally opposite corners of the rectangle.

READ [REA] Read DATA values

READ v_name

Reads the relative data items and places them in the given variable. v_name is the name of a variable which the read data should be placed in. As is described in RESTORE, a pointer can point to a label. READ reads the data info following this label and places it in the variable. If RESTORE is not used, the first data line in the program is read. If there are fewer data items then READ uses, the appropriate error message is displayed.

RELSEEK [REL] — Move file pointer

```
RELSEEK #Channel_number, Byte_count
```
(Towards end of file)

```
RELSEEK #Channel_number, -Byte_count
```
(Towards beginning of file)

RELSEEK moves the file pointer to the given file a number of bytes either towards the beginning of the file or towards the end of the file. Channel_number is the indentification number of the file whose pointer should be moved. Negative values indicate that the pointer should be moved towards the beginning of the file, positive values mean towards the end of the file. It is important to note that the pointer is not moved past the end of the file, nor before the beginning of the file.

REM [R or '] — Insert comments

```
REM
REM Comment
```

Any desired comment line can be placed in the program.

REPEAT...UNTIL [REP...U] Conditional loop

```
REPEAT
...program commands...
UNTIL condition
```

Sets up a loop with the exit condition at the end of the loop. The exit condition is tested at the end of each execution of the loop. That means that the loop is always executed at least one time. If the exit condition is true, program execution continues at the line following the UNTIL command. REPEAT...UNTIL loops can be nested as deep as desired.

RESERVE Set BASIC memory size

```
RESERVE number
```

Sets the amount of memory available for BASIC (FRE(0)). number is the new size of the BASIC working memory in bytes. If BASIC memory is reduced in size, the free area above the new end of memory can be used for other purposes. Commands like BMOVE, BLOAD and BGET (and also ".RSC" files) can then be used. If BASIC memory is increased, it should be noted that GEM and GEMDOS access a memory area directly below the screen memory (XBIOS(2)) for various processes (alert boxes, file selector box, DTA etc.). The interpreter automatically sets the upper limit of the free BASIC memory to 16384 bytes below the video RAM. It would be possible to use all of the RAM up to the video RAM (RESERVE FRE(0)+16384). In this case HIMEM and XBIOS(2) would be identical. But if this is done, then GEM is no longer able to do things like display file selector boxes. The

memory should therefore be configured such that at least 4000 bytes below XBIOS (2) is reserved for GEM.

If the memory above the BASIC storage is used up to the screen boundary, GEM overwrites data within 4000 bytes of the screen start. To prevent GEM access to the reserved area above BASIC, a memory area can be reserved with the GEMDOS function $48 (Malloc). Memory allocated in this manner is released with another GEMDOS function, $49 (Mfree).

Release of memory with function $49 is also necessary when a program is loaded with the EXEC 3... command and is not started, so that it does not release the memory occupied by it again.

RESTORE [RES] — Set DATA pointer

RESTORE
(sets the data pointer to the very first data line)

RESTORE label
(set the data pointer to the line indicated with LABEL:)

RESTORE restores the data memory or sets the data pointer to the given label. RESTORE without input of a label name indicates that the read-data pointer is set to the very first data line in the entire program. All following READ uses data items one after another, starting at the first DATA line. If the command is given with a label name, the data pointer is set to the first data line following the given label. From there the desired data lines are read. There must be enough data lines from this point to the end of the program for all READ statements otherwise the appropriate error message is displayed.

RESUME [RESU] Continue program execution

```
RESUME
```

After an error handling routine, this command repeats the line in which the error occurred.

`RESUME NEXT`
After an error handling routine, continuing at the line following the error.

`RESUME LABEL`
After an error handling routine, the program line which follows the given label.

Sets where the program continues execution after the execution of an error handling routine. If `RESUME` is used without any parameters, the program continues at the line where the error occurred, repeating execution of this line. `RESUME NEXT` indicates that program execution continues at the line following the error. If the given label is outside of the error routine, the `GOSUB` register is erased and all global variables are restored. With the bomb error (102-109), `RESUME` and `RESUME NEXT` cannot be used. In this case, `RESUME LABEL` must be used.

RETURN [RET] Procedure end

```
RETURN
```

Marks the end of a procedure. The procedure is exited and execution continues at the line following the procedure call.

RIGHT$ Return right justified characters

```
RIGHT$(string,count)
```

Returns a certain number of characters of a given string beginning with the last character of the string. `string` is a string variable or expression, followed by a comma and then the number of `count` characters to read from the string. `count` can also be a variable. If `count` is not given, the last character of the string is returned. If `count` is larger then the length of `string`, the entire string is returned. In the case when `string` is null (""), a null string is returned. `RIGHT$` can be displayed directly, placed in a string variable or used in a string expression.

RMDIR [RM] Delete a directory

```
RMDIR "directory name"
```

`RMDIR` deletes the given directory. `directory name` is the name of the directory to delete. Before a directory is deleted, all files are deleted. The directory must be completely empty.

RND — Random decimal number (0<x<1)

```
RND (Dummy)
RND
```

Returns a random number between 0 and 1. The Dummy argument need not be used. A random number with 11 decimal places is generated within the range 0(inclusive) and 1(exclusive). For example: 0.57964183275.

RSET [RS] — Set a right justified string

```
RSET string_var$=string expression
```

Places a given string right justified in a string variable. string_expression can be any desired character string or string variable. The contents of this string are then placed right justified in string_var$. The length of string_var$ is never changed. If the length of string_var$ is smaller then the length of string_expression, the string expression is truncated to the length of string_var$. If string_var$ is longer then the string expression, the beginning of the string_var$ is filled with blank spaces.

RUN [RU] — Start program

RUN

The RUN command initializes and starts the BASIC program in memory from the first line. At the same time, all variables and the monitor are cleared. RUN can be in the program or be given in direct mode.

SAVE [SA] — Save program (in coded form)

SAVE "Program name"

This command saves the work memory under the given name in a coded form on the diskette and is identical to the SAVE menu command. Program name specifies the name for the file to be saved. If no extension is given, the interpreter automatically supplies .BAS. If a program with the same name is already on the diskette, the SAVE command overwrites the old program.

SEEK [SEE] Set file pointer

```
SEEK #Channel_number, location
```
(from beginning of file)

```
SEEK #Channel_number, -location
```
(from end of file)

The file pointer of a file can be set to any desired location in the file. `Channel_number` is the file identifier (0-99) of the file whose pointer is set. This value is less than or equal to the size of the file. A negative number indicates counting from the end of the file. (e.g. `SEEK #0, -1` places the pointer at the next to last byte of the file).]

SETCOLOR [SE] Set color register

```
SETCOLOR register, red_portion, green_portion,
blue_portion
SETCOLOR register, mix_value
```

With this command the color tone (RGB mix) for a single color register can be set. The register can be set by specifying the amount of each color (RGB = 0 to 7), or with a `mix_value` (1 to 1911). In the second case, the value is determined as follows:

```
red_portion      * 256
+ green_portion * 16
+ blue_portion
--------------------
= mix_value
```

Color registers with special meaning:

Register 0 = background color
Register 1 = print output color
Register 2 = text color of the BASIC editor

SETTIME [SETT] — Set clock time and date

```
SETTIME new_time$,new_date$
```

Sets the internal clock to a new time and date. The time and date specifications updated with `SETTIME` can be read from within a program with `time$` and `date$` and are also available on the system level (TOS, control panel accessory...). The inputs `new_time$` and `new_date$` can be passed directly as constant strings enclosed in quotation marks or as string variables. Both strings must be passed.

Format:

`new_time$`	"hh:mm:ss" or "hhmmss"
`new_date$`	"dd.mm.yyyy" or "dd.mm.yy"

In the `new_time$` string hh stands for hours, mm for minutes, and ss for seconds, each two digits. The separators (colons) are optional. In the `new_date$` string dd stands for the day, mm for the month, and yyyy or yy for the year. The separators (periods) are not optional. The year can be specified as two digits if it falls between 1980 and 2079. The inputs are checked for plausibility.

SGET — Store screen

```
SGET var$
```

Transfers the current screen memory to a string variable. `var$` is the name of the string variable in which the screen contents are stored. In contrast to `GET X0,Y0,X1,Y1,A$`, this command doesn't have to work with divided words. This means that it is about 15 times faster than `GET`, but it also means that the variable contents have a different format. A screen read with `SGET` cannot be manipulated with `PUT`. This command is the reverse of `SPUT`.

It is functionally identical to:

```
A$=space$(32000)
Bmove Xbios(2),varptr(A$),32000
```

SGN — Determine sign

```
SGN(x)
```

Returns the sign of a number. x is a numeric expression of which the sign is found. The function returns:

+1	when x>0	(positive)
-1	when x<0	(negative)
0	when x=0	(zero)

SHOWM — Enable mouse pointer

```
SHOWM
```

A mouse pointer disabled with HIDEM can be turned back on with this command, with VDISYS 122 or by returning to the editor.

SIN — Sine

```
SIN(x)
```

Calculates the sine of an angle given in radians. The sine of an angle is defined for right triangles as the length of the opposite side divided by the length of the hypothenuse. The x argument is an angle expressed in radians for which the sine is calculated. If this value is given in degrees, it must first be multiplied by PI/180.

SOUND [SO] — Output a tone

```
SOUND channel, volume, note, octave, duration
SOUND channel, volume, #period, duration
```

The SOUND chip has the capabilities of 3 channel output. This means that 3 different tones or noises can be played at the same time. To do that, the SOUND Channel must first be initialized

by the WAVE command. The SOUND command uses 4 or 5 parameters:

`channel` The number of the output channel (1, 2 or 3).

`volume` Sets the volume of the output from 0(quiet) to 15(loud).

There are two methods of setting the tone:

a) Chromatic

Note: Set the "musical pitch" for the sound output:

1	2	3	4	5	6	7	8	9	10	11	12
C	C# (Db)	D	D# (Eb)	E	F	F# (Gb)	G	G# (Ab)	A	A# (Bb)	B

`octave` Sets which octave the tone is played in. A value of 1 selects the lowest, 8 the highest.

b) Numeric:

`period` Sets the "physical pitch" for the sound output. It is a value, proceeded by "#", which is the maximum output frequency (12500 Hertz) divided by the desired frequency (a whole number): `#Period = TRUNC (125000 /FREQUENCY + 0.5)`

A note can be played by using two different methods. For example, concert A, for 2 seconds:

```
SOUND 1,15,10,4,100     (10th note, 4th octave)
SOUND 1,15,#284,100     (TRUNC(125000 Hz / 440 Hz + 0.5)
```

`duration`	Sets how much time passes before the next command is executed. The tone is played for the given duration (in 1/50 second steps, i.e 100 = 2 seconds).

SPACE$ — Set a string to blanks

```
SPACE$(count)
```

Outputs or places in a string expression a desired number of blank spaces. The `count` argument is the number of blanks to be output. It cannot be negative and not larger than 32767 (the maximum string length). The blank spaces from `SPACE$` can be directly output, placed in a variable or used in a string expression.

SPC — Output blank spaces

```
SPC(count)
```

This command is used along with the `PRINT` command to output `count` spaces. It cannot be integrated into string constructions. Within a `PRINT` output, any desired number of spaces can be output.

S/SD/SLPOKE [SP, SD, SL] Supervisor poke

SPOKE address, byte
(writes one byte)

SDPOKE complete_address, word
(writes two bytes)

SLPOKE complete_address, long
(writes four bytes)

Places the given value, at a privileged memory location in the given format in supervisor mode. This command is used exactly like POKE, DPOKE or LPOKE. The difference is that "privileged" memory can be accessed and the contents changed (supervisor mode). These commands are inverses of PEEK, DPEEK, and LPEEK in supervisor mode.

SPRITE [SPR] Set and erase sprites

SPRITE Spr_string$,x,y
(set sprite)

SPRITE Spr_string$
(erase sprite)

Software sprites can be generated and either placed or erased from the screen. The sprite definition is the same as the mouse pointer in DEFMOUSE. All data are two byte words in MKI$ format in a string variable. The sprite pattern is 16 lines by 16 columns.

Word 1 X-coordinate of the action point in the sprite form.

Word 2 Y-coordinate of the action point in the sprite form.

```
S.pr$ = S.pr$ + MKI$(1) + MKI$(1)          = upper left
S.pr$ = S.pr$ + MKI$(1) + MKI$(65535)      = lower left
S.pr$ = S.pr$ + MKI$(65535) + MKI$(1)      = upper right
S.pr$ = S.pr$ + MKI$(65535) + MKI$(65535)  = lower right
```

Word 3 Mode (normal = MKI$(0), XOR = MKI$(1))

Word 4 Mask color (background of the sprite picture).White = MKI$(0), black = MKI$(1)

Word 5 Cursor color (sprite)

Word 6 to 37
Alternate lines of the sprite mask and the sprite form, beginning with the sprite mask.

With the input of such a sprite definition string, is the coordinate where the sprite should be displayed. Calling the command again with the same sprite but different coordinates erases the previous sprite. If the sprite should be completely removed from the screen, the coordinates are not given in the command.

SPUT — Display screen

```
SPUT var$
```

Transfers the contents of a string variable to the start of the screen memory. `var$` is the name of a variable whose contents are displayed. This does not necessarily have to be a variable initialized with `SGET`. If the variable contents are greater than 32000 bytes, only the first 32000 bytes are used. This command is the complement of `SGET`. It is considerably faster than `PUT 0,0,A$`.

SQR — Square root

```
SQR(x)
```

Calculates the square root of a number. The `x` argument is a numeric expression which cannot be negative. The function only returns the square root, if a higher root is needed, the exponential function must be used with exponents: x^1/3, x^1/4,...

STOP [ST] — Halt program

```
STOP
```

With the `STOP` command, program execution can be halted at any desired location. Variables are not erased, files are not closed (as with `END`) and the program can be started again with the direct mode `CONT` command. (See `CONT`).

STR$ — Numeric -> string

```
STR$(value)
```

A numeric value is converted into a string. `value` can be any desired value in any desired number system. With a hexadecimal, binary or octal value, the number is converted first to decimal. The returned string has the same length as the number of digits of the given value in decimal format. The return value is no longer a

number, rather a single character string containing digits. This value can be directly output, placed in a string variable or used in a string expression. `STR$` is the inverse of the `VAL` function.

STRING$ — Output a string

`STRING$(count, string_expression)`
(with characters)

`STRING$(count, ASCII)`
(with ASCII code value)

Constructs a character string which contains as many occurrences as desired of a given string. The `count` parameter sets how many times `string_expression` is used. `string_expression` can either be normal text or a string variable. A second variation allows the use of ASCII code values (e.g. for A, ASCII=65). The ASCII values must be positive whole numbers less then 256 (ASCII MOD 256). The maximum length of the output is 32767 characters (maximum string length). If this limit is exceeded, an error window is displayed.

SUB — Subtraction command

`SUB v_name1, v_name2`
`SUB v_name1, const`

Subtraction of variables with result assignment. `v_name1` and `v_name2` can be numeric variables or array varriables. `const` is a numeric constant. The value after the comma is subtracted from

the value before the comma and the result is placed in the variable v_name1.

SWAP — Swap values

```
SWAP v_name1, v_name2
```

Any two numeric, alphanumeric or boolean variables and arrays can be swapped as long as v_name1 and v_name2 are of the same type. With arrays, the dimensions are also swapped.

SYSTEM [SYS] — Leave interpreter

```
SYSTEM
```

SYSTEM is identical to QUIT and means that the program ends. The interpreter is exited and control is returned to the desktop. There is no verification question before this happens.

TAB — Tab to a position

```
TAB(Position)
```

A tab position is set which can be used with the next PRINT statement. Position is a value in the range 0 to 255. Larger numbers are reduced to value MOD 256. If the cursor is in a line

after the last given tab position and the value of the tab position is between the current cursor position and 256, the tab is in the same line. If the tab value is smaller then the current cursor position, the tab is in the next line. The TAB command can only be used in combination with the PRINT command. It is not possible to use in a string output.

TAN — Tangent

TAN(x)

Calculates the tangent of an angle given in radians. The tangent of an angle is defined for right triangles as the length of the opposite side divided by the length of the adjacent side. The x argument is an angle expressed in radians for which the tangent is calculated. If this value is given in degrees, it must first be multiplied by PI/180. The inverse function is ATN.

TEXT [T] — Display text in graphic mode

TEXT x,y, "text"
(text)

TEXT x,y, string_expression
(text together)

TEXT x,y, v_name$
(a string variable)

TEXT x,y,length, v_name$
(variable character spacing)

```
TEXT x,y,-length, v_name$
```
(variable word spacing)

Graphic text is placed at any desired screen location. The output text can be given directly as text (`"text"`) or as a string expression (`A$ + str$(1)`). The text is placed at the screen location (`x,y`). Text attributes are set with the `DEFTEXT` command. `x,y` is the point coordinate at which the text is displayed left justified. The `length` parameter is optional. If length is given for the text, the text is placed so that it fills this space. This occurs by stretching or shrinking the letter spacing (positive `length`) or word spacing (negative `length`).

TIME$ — Return system time

```
TIME$
```

Returns a string containing the current system time. `TIME$` can be printed directly, assigned to a string variable, used in a string expression or conditional statement. The output format is "hh:mm:ss". The seconds are incremented in steps of two.

TIMER — Determine running time

```
TIMER
```

The time counter of the ATARI ST is set to zero when the computer is turned on, and then incremented 200 times each second. It is then possible to calculate the time between

operations. This number can be directly displayed, placed in a variable or used in a condition statement.

TITLEW [TIT] Set window title line

```
TITLEW handle,string_expression
```

Sets or changes the title of a GEM window. A window can be given a title which appears in an opened window within the grey move bar. The TITLEW command works just like the INFOW command.

Exception:

```
string_expression = " " (one space) -> no title

string_expression = "" (null string) -> window
can no longer be moved
```

TROFF Turn off debugging

```
TROFF
```

TROFF ends the program listing which began with TRON. Using TROFF and TRON, any desired portion of the program can be watched during its execution. After this command, the program is again in normal mode. If the program lines are written to a file with TRON #channel_number, the input of a channel number can be left off in the TROFF command.

TRON — Turn on debugging

```
TRON
TRON #Channel_number
```

Lists the program lines as they are executed on the monitor, printer or diskette file. If the command is used without the `#Channel_number` option, the current program is listed on the monitor. `Channel_number` contains the number of the open file (0-99) which the listing should be written to. This file must already have been opened with `OPEN` (see `OPEN`).

TRUE — True constant (-1)

```
TRUE
```

Contains the constant -1. A boolean value is returned by various functions (such as `EXIST`), and for better readability, this constant can be used in conditions instead of the value -1 (see also `FALSE`).

TRUNC — Integer function

```
TRUNC(x)
```

Returns the whole (integer) portion of a real number by truncating the portion right of the decimal point. `x` is any desired numeric

expression. The function rounds numbers neither up or down, rather only removes the decimal portion. The fractional portion of a number is returned with FRAC. TRUNC is identical to FIX.

TYPE — Return variable type

```
TYPE(pointer)
```

Returns the type of the variable to which pointer points. Pointer is an integer that points to the desired variable (see *). The variable can also be specified directly with an asterisk in front of it. The function returns a value from -1 to 7:

-1	Error occurred
0	Real variable (var)
1	String variable (var$)
2	Integer variable (var%)
3	Boolean variable (var!)
4	Real array variable (var())
5	String array variable (var$())
6	Integer array variable (var%())
7	Boolean array variable (var!())

UPPER$ — Change letters from lowercase to all caps

```
UPPER$(string_expression)
```

The character string string_expression can either be text or a string variable. The UPPER$ function converts all lowercase letters (ASCII values 97 to 129) to uppercase letters (ASCII

values 65 to 90). All other alphanumeric characters remain unchanged. The control characters (ASCII values 0 to 31) cannot be converted.

VAL — String -> numeric

```
VAL(string_expression)
```

Converts all digits at the beginning of a string to a decimal real number. `string_expression` is any desired character string or string variable which is converted. The beginning of the string can be a prefix specifying another number system (&X, &O, &H). The conversion ends when the end of the string is reached or a non-digit is found, whichever occurs first. The returned value is always a decimal real number, regardless of the input form. If the first character of the string is a non-numeric text character (something other than 0...9 or &), or the string is empty, zero is returned. The returned value can be directly output, placed in a variable or used in a condition statement.

VAL? — Number of numeric text characters

```
VAL?(string_expression)
```

Returns the number of characters, starting from the beginning of the string, which can be converted to a numeric value. `string_expression` is any desired string expression or string variable. The count continues only until a non-convertible character is found. The returned value can be directly output, placed in a variable or used in a condition statement.

VARPTR — Determine variable address

```
VARPTR(V_name)
```

Returns the address of a numerical variable or the address of the first character of a string variable. `V_name` stands for any variable.

VDIBASE — Return base of VDI array

```
VDIBASE
```

This system variable contains the base address of the VDI parameter block. This is a memory block of about 300 bytes which GEM uses for managing the various VDI parameters. This includes information about the type size, graphics mode, etc., as well as data for managing the VDI environment (open/close work station, fonts, screen organization, etc.). Since there is currently no documentation from Atari about this area, we can't say anything about what is stored here. `POKE`ing into this area of memory can change various VDI parameters, but it can also cause the computer to crash. Above the BASIC interpreter GfA BASIC manages the various tables and program variables. `VDIBASE` comes after this area. After the VDI range comes a variable list accessed by the interpreter.

This program lists the contents of the VDI block:

```
Print "BYTES:"
For I=0 To 300
  Print Peek(Vdibase+I)'
Next I
```

```
Print Chr$(10);Chr$(13);"WORDS:"
For I=0 To 300 Step 2
  Print Dpeek(Vdibase+I)'
Next I
Print Chr$(10);Chr$(13);"LONGS:"
For I=0 To 300 Step 4
  Print Lpeek(Vdibase+I)'
Next I
Void Inp(2)
```

VDISYS — Call VDI routines

VDISYS
(repeat last opcode)

VDISYS opcode
(new opcode)

This command allows VDI system routines to be called. The VDI (Virtual Device Interface) supports graphics and text output and settings. opcode contains the function number of the GEM routine to be called. The GEM parameter arrays must be prepared before the call. If you just change the array contents and call the same function again, you can omit the opcode specification. The arrays to prepare are listed under "System data and addresses" with the suffix of VDI.

```
Dpoke Contrl+2,4
Dpoke Contrl+6,2
Dpoke Contrl+10,3
Dpoke Contrl+12,2
Dpoke Intin,1600
Dpoke Intin+2,2000
Dpoke Ptsin,580
Dpoke Ptsin+2,200
```

```
Dpoke Ptsin+12,490
Vdisys 11
For I=1 To 4
  Deffill ,2,I
  Vdisys
Next I
```

VDISYS
Current Fill Area Attributes

Graphics attributes

Several current graphic settings are returned. Values between 0 and 1000 are returned in three component variables, depending on the color.

```
Dpoke Contrl+2,0
Dpoke Contrl+6,0
Vdisys 37
Fill_type%=Dpeek(Intout)
Fill_color%=Dpeek(Intout+2)
Fill_pattern%=Dpeek(Intout+4)
Graf_mode%=Dpeek(Intout+6)
Border%=Dpeek(Intout+8)
```

VDISYS
Inquire Color Representation

Color mixing

This function reads the individual color components (RGB) of a given object or the current object color.

```
Dpoke Contrl+2,0
Dpoke Contrl+6,2
```

```
Dpoke Intin,color index or register
Dpoke Intin+2,Flag (current color=1 / other=0)
Vdisys 104
Colorindex%=Dpeek(Gintout)
Red_comp%=Dpeek(Gintout+2)
Green_comp%=Dpeek(Gintout+4)
Blue_comp%=Dpeek(Gintout+6)
```

VDISYS
Set Character Cell High — Set text size

Allows various text sizes to be set. With `Handle% 2` this function does the same thing as the corresponding `DEFTEXT` parameter. Various AES processes can also be affected by passing a `Handle%` of 1. The value to be passed in `T.high%` can be any valid `DEFTEXT` value.

```
Dpoke Contrl+2,0
Dpoke Contrl+6,1
Dpoke Contrl+12,Handle%
Dpoke Intin,T.high%
Vdisys 107
```

VDISYS
Set Clipping Rectangle — Define graphics region

All graphic outputs (`PBOX`, `LINE`, etc.) following this function are clipped to fit in the screen region defined here. Block commands like `PUT` and `SPUT` are not affected.

```
Dpoke Contrl+2,4
Dpoke Contrl+6,1
```

```
Dpoke Intin,Flag (clipping on=1 / off=0)
Dpoke Ptsin,X-coord of the upper left corner
Dpoke Ptsin+2,Y-coord of the upper left corner
Dpoke Ptsin+4,X-coord of the lower right corner
Dpoke Ptsin+6,Y-coord of the lower right corner
Vdisys 129
```

VDISYS
Set Fill Perimeter Visibility
Box borders

The PBOX, PRBOX, PELLIPSE, PCIRCLE and POLYFILL commands surround the filled surfaces which they display with a line drawn in the object color. This border can be turned on or off.

```
Dpoke Intin,Flag (border off = 0 / on = 1)
Vdisys 104
```

VDISYS
Set graphics text special effects
Set text style

Allows various text styles to be set. Although this function has the same purpose as the second parameter of the BASIC command DEFTEXT, we mention it here because by passing a Handle% of 1 the text style of the desktop, alert and file selector boxes, or of dialog boxes can be changed. The value passed in T.style% is identical to the values explained for DEFTEXT.

```
Dpoke Contrl+2,0
Dpoke Contrl+6,1
Dpoke Contrl+12,Handle%
```

```
Dpoke Intin,T.style%
Vdisys 106
```

VOID — Function call without return value

```
VOID function(parameter)
```

Calls a function without using its return value. In many cases it is unnecessary for functions to return values because return values are often placed in certain of the parameters. To save space (and time), these functions can be called with VOID.

```
instead of:  A=INP(2)              !wait for a
                                   key
==>          Void INP(2)

instead of:  A=Fre(0)              !garbage
                                   collection
==>          Void Fre(0)

instead of:  A=Gemdos(&HE,1)       !call disk B
==>          Void Gemdos(&HE,1)
```

VSYNC — Wait for vertical blank

```
VSYNC
```

VSYNC waits for the next vertical blank to occur. For graphics output like SPUT or PUT it can be useful to wait for the next screen construction. By waiting for vertical synchronization,

flickering can be eliminated. Graphic operations which take longer than a computer requires to construct the screen (monochrome = 70Hz) always encounter interference.

WAVE [WA] — Initialize sound register

WAVE channel, envelope, c_form, period, duration

WAVE enables access to the registers of the sound chip. The sound influence and the voices can then be set. WAVE must be used to activate a channel of the sound chip before the SOUND command can be used. channel sets the sound chip channel to be initialized. SOUND uses this value, which is added bitwise as follows:

1 =	Bit 0	(Channel 1)
2 =	Bit 1	(Channel 2)
4 =	Bit 2	(Channel 3)
8 =	Bit 3	(noise added to Channel 1)
16 =	Bit 4	(noise added to Channel 2)
32 =	Bit 5	(noise added to Channel 3)

envelope sets which channel (including noise) is modulated (changed) by the envelope. This value is also set bitwise in the same way as channel. The curve_form parameter defines the envelope type:

Linear fall	= 0, 1, 2, 3, 9
Linear fall, erratic	= 11
Linear rise, breaking	= 4, 5, 6, 7, 15
Linear rise, stopping	= 13
Sawtooth fall	= 8
Sawtooth rise	= 12
Triangle, initial fall	= 10
Triangle, initial rise	= 14

period — A numerical value which sets how long the envelope is set. The smaller it is, the more the envelope is stretched (graphical representation), and the modulation speed is raised.

duration — Sets how much time you must wait before the next command is executed. When a sound is activated, the sound chip turns on. Using the WAVE 0,0 command the sound continues until a key is pressed.

WHILE...WEND [W...WE] — Conditional loop

```
WHILE condition
...program commands...
WEND
```

The loop exist condition is tested at the beginning of the loop. That means that the loop is not executed, if the condition is false. When the loop is exited, execution continues at the line following the WEND command. WHILE...WEND loops can be nested as deep as desired.

WINDTAB — Window management table

```
DPEEK (WINDTAB)
DPOKE WINDTAB,value
```

Start address of a memory area from which data for managing GEM windows can be read, or whose contents can be changed if

necessary or desired. The window table `Windtab` is a reserved variable. 16-bit values are stored in the table which contain information about the current state of the windows.

The `WINDTAB` table has the following construction:

`Windtab`	=	Window handle 1
`Windtab+2`	=	Attributes for window 1
`Windtab+4`	=	X-coordinate for window 1
`Windtab+6`	=	Y-coordinate for window 1
`Windtab+8`	=	Width of window 1
`Windtab+10`	=	Height of window 1
`Windtab+12` to `Windtab+22`	=	Corresponding specifications for window 2
`Windtab+24` to `Windtab+34`	=	Corresponding specifications for window 3
`Windtab+36` to `Windtab+46`	=	Corresponding specifications for window 3
`Windtab+52`	=	Screen X coordinates
`Windtab+54`	=	Screen Y coordinates
`Windtab+56`	=	Screen width
`Windtab+58`	=	Screen height
`Windtab+60`	=	X-intersection of the four windows
`Windtab+62`	=	Y-intersection of the four windows
`Windtab+64`	=	X-origin for graphics commands
`Windtab+68`	=	Y-origin for graphics commands

In addition, four words (16-bits each) of this table can be used to determine which border objects are active for each window. This is done through:

```
Dpoke Windtab+2,  12-bit value for window 1
Dpoke Windtab+14, 12-bit value for window 2
Dpoke Windtab+26, 12-bit value for window 3
Dpoke Windtab+38, 12-bit value for window 4
```

A 12-bit value is passed to these WINDTAB locations. The components can be determined by individual settings or addition of the bit values:

1 =	The NAME
2 =	The upper left CLOSE field
4 =	The upper right FULL field
8 =	The MOVE bar at the top
16 =	The info line under the MOVE bar
32 =	The lower right SIZE field
64 =	The right up arrow
128 =	The right down arrow
256 =	The right VSLIDE bar (vertical scroll bar)
512 =	The lower left arrow
1024 =	The lower right arrow
2048 =	The lower HSLIDE bar (horizontal scroll bar)

WRITE — Output data

```
WRITE
WRITE "Text"
WRITE "Text", "Text", V_name, V_name$, output;
```

WRITE is able to display text or data on the monitor and is very similar to PRINT, but with a different syntactical structure. The output characters are separated from each other in the list with commas. The suppressing of CR and LF is accomplished by ending the line with a semicolon. WRITE without any output list displays a blank line in the same fashion as the PRINT command. Further uses of this command are under WRITE#.

WRITE# [WR] — Write sequential data

```
WRITE#Channel_number
WRITE#Channel_number, "Text"
WRITE#Channel_number, "Text", V_name, V_name$...
```

The main use of this command is saving data that can be later read with the INPUT# command. If the INPUT# command is to read several data items at once, each must be separated by a comma in the WRITE# command. Channel_number is the file number (0-99) to which the data should be written. Text, V_name and V_name$ are any desired variables or strings, each separated with a comma.

XBIOS — Call TOS routines

```
XBIOS (opcode, parameter_list
XBIOS= Extended Basic Input/Output System
```

The operating system routines offered by the various subsystems can be called with three commands, BIOS, XBIOS and GEMDOS. The Atari ST has three different system levels, each of which has special tasks. For example, disks can be formatted, font tables can be loaded, and so on by specifying the appropriate opcode (function number). The number of parameters passed depends on the system routine called. If long words are expected by the function (such as for address parameters), the corresponding value should be prefixed with L:. If no prefix is present, the passed values are interpreted as 16-bit values. If strings are passed, the start address must be specified (see also VARPTR). A numerical variable can be specified to receive the result of a parameter returned to the program.

XBIOS
FLOPFMT — Format track

```
A$=Space$(8000)
Addr%=Varptr(A$)
A%=Xbios(10,L:Addr%,L:1,Dev%,Spt%,Trk%,Side%,1,L
:&H87654321,$HE5E5)
```

This function formats a single track. A$ is used as a buffer to hold the track data to be written (at least 8000 bytes).

Spt% = sectors per track (usually 9)

All other parameters have the same meaning as those described for FLOPRD. Upon return, the track data is contained in A$ and a return code is placed in A% which has the same meaning as those described for FLOPRD. To completely format a disk, a boot sector must be created (too involved a subject for this book). With this function alone, only single tracks can be formatted.

XBIOS
FLOPRD — Read sector(s)

```
A$=Space$(512*Num%)
Addr%=Varptr(A$)
A%=Xbios(8,L:Addr%,L:1,Dev%,Sec%,Th%,Side%,Num%)
```

One or more sectors can be read from disk. A$ is used to prepare a buffer to store the bytes read.

`Num%` Number of sectors to read (1-9)
`Addr%` Address of the buffer (here Varptr(A$))
`Dev%` Number of the desired drive (0=A; 1=B)
`Sec%` Number of the first sector to be read (0-8)
`Th%` Number of the track containing the sectors to be read (0-79/81)
`Side%` Side of the disk on which the desired track is located (0 = single-sided; 0/1 = double-sided)

Upon return the bytes read can be displayed with `PRINT A$`. A code is placed in `A%` by the function to indicate the result of the operation.

Meaning:

0	No error occurred
-1	General error
-2	Drive not ready
-3	Unknown command
-5	Illegal command
-6	Track not found
-7	Boot sector invalid
-8	Sector not found
-10	Write error
-12	Same as -1
-13	Disk is write-protected
-14	Disk was changed
-15	Device unknown
-16	Sector contains errors
-17	Disk not in drive

XBIOS
FLOPVER Verify sector(s)

```
A$="contains the characters to be compared"
Addr%=Varptr(A$)
A%=Xbios(19,L:Addr%,L:1,Dev%,Sec%,Th%,Side%,
Num%)
```

One or more sectors on the disk can be compared with the contents of a buffer. A$ represents a buffer containing the data to be compared with those written to the disk. It must be 512 times as large as the number of sectors to be compared (Num%). Usually this function is used to determine whether the bytes transferred with FLOPRD or FLOPWR were read/written correctly. The parameters to pass have the same meanings as those for FLOPRD. The return value in A% is also the same.

XBIOS
FLOPWR Write sector(s)

```
A$="contains the data to be written"
Addr%=Varptr(A$)
A%=Xbios(9,L:Addr%,L:1,Dev%,Sec%,Th%,Side%,Num%)
```

This function writes one or more sectors to disk. A$ represents a buffer containing the data to be written. It must be 512 times as large as the number of sectors to be written (Num%). The parameters passed to the function have the same functions as those described for FLOPRD. The return value has the same meaning as that of FLOPRD.

XBIOS
KBRATE Set keyboard repeat rate

```
A%=Xbios(35,Delay%,Rep%)
```

Sets reaction delay and repeat rate for the keyboard repeat function. `Delay%` contains the delay value while `Rep%` contains the repeat rate. The value 1 is a very fast rate, and the larger this value, the slower the rate. If the value 0 is passed for `Rep%`, then repeat is turned off. To leave a parameter unchanged, pass the value -1. Upon return, a two-byte value is stored in `A%`. The high byte contains the previous repeat rate and the low byte the previous delay value.

XBIOS
LOGBASE Return logical screen address

```
A%=Xbios(3)
```

The logical base address of the screen memory is returned. The address is in `A%` upon return.

XBIOS
MIDIWS Send string to MIDI

```
A$="MIDI string to send"
Void Xbios(12,Len(A$)-1,L:Varptr(A$))
```

This function sends a string to the MIDIOUT port.

XBIOS
PHYSBASE Return physical screen address

```
A%=Xbios(2)
```

The physical base address of the screen memory is returned. Upon return, this address is available in `A%`.

XBIOS
PUNTAES Turn AES off

```
Void Xbios(39)
```

If the AES is in RAM, it is disabled when this function is called.

XBIOS
SETCOLOR Read/set color registers

```
A%=Xbios(7,Reg%,Col%)
```

Individual colors can be set or read with this function. The `set color` variant is identical to the `SETCOLOR` command. The current color value of the specified color register can also be read. `Reg%` contains the desired register number and `Col%` is the color value to be set (0-1911). To read the current color value, pass the value -1 in `Col%`.

XBIOS
SETPRT — Set/read printer attributes

```
A%=Xbios(33,Attr%)
```

With this function, various printer settings can be read or set. If the attributes are set, a 16-bit value is passed in `Attr%`. If the value -1 is passed as `Attr%`, the current settings are returned by the function in `A%`. The first six bits have the following meanings:

	Set	Unset
Bit 1	Dot-matrix	Daisy wheel
Bit 2	Color printer	Black/white printer
Bit 3	Atari	Epson
Bit 4	Draft	Final
Bit 5	Centronics	RS232
Bit 6	Continuous	Single-sheet

Bits 7-16 are not of interest. This function allows you to make the same settings as those made with the Printer interface accessory.

XBIOS
SETSCREEN — Set screen parameters

```
Void Xbios(5,L:L.ad%,L:P.ad%,Res%)
```

The resolution as well as the logical and physical addresses of the screen can be changed with this function.

`L.ad%` contains the new Logbase
`P.ad%` contains the new Physbase
`Res%` contains the Resolution

To leave a parameter unchanged, pass a -1 in place of a new value.

Error Messages

Editor Error Messages

WHILE without WEND
REPEAT without UNTIL
DO without LOOP
FOR without NEXT
WEND without WHILE
LOOP without DO
NEXT without FOR
IF without ENDIF
ENDIF without IF
ELSE without IF
ELSE without ENDIF
EXIT without LOOP
PROCEDURE without RETURN
PROCEDURE in LOOP
MULTIPLY defined PROCEDURE
MULTIPLY defined LABEL
LOCAL only in PROCEDURE
LOCAL not in LOOP
MULTIPLY defined FUNCTION
GOTO into/out of FOR...NEXT
GOTO into/out of PROCEDURE
RESUME in FOR...NEXT LOOP
RESUME without PROCEDURE
Syntax error
Line too long

BASIC Error Messages

Error	Text
0	Division by zero
1	Overflow
2	Number not integer - 2147483648...2147483648
3	Number not byte 0...255
4	Number not word 0...65535
5	Square root only for positive numbers
6	Logarithms only for numbers > 0
7	Unknown error
8	Out of memory
9	Function or command not yet possible
10	String too long. Max 32767 characters
11	Not a GfA BASIC V2.0 program
12	Program too long, memory full. New
13	Not GfA BASIC. Program file too short. New
14	Array redimensioned
15	Array not dimensioned
16	Array index too large
17	Dim index too large
18	Wrong number of indices
19	Procedure not found
20	Label not found
21	Only allowed for open: "I"NPUT, "O"UTPUT, "R"ANDOM, "A"PPEND, "U"PDATE
22	File already open
23	Wrong file #
24	File not open
25	Illegal input. Not a number
26	End of file reached
27	Too many points for POLYLINE/POLYFILL. Max 128
28	Array must be one-dimensional
29	Number of points is larger than array

30	Merge - not an ASCII file
31	Merge - file too long - terminating
32	==> syntax not correct
33	Label not defined
34	Too little data
35	Data not numeric
36	Syntax error in data. Use "" pairwise
37	Disk full
38	Command not allowed in direct mode
39	Program error. No GOSUB possible
40	Clear not allowed in FOR...NEXT loops or Procedures
41	Cont not possible
42	Too few parameters
43	Expression too complex
44	Function not defined
45	Too many parameters
46	Incorrect parameter. Must be a number
47	Incorrect parameter. Must be a string
48	Open "R" record length incorrect
50	Not a random file
51	Only one field allowed per open "R"
52	Fields larger than record length
53	Too many fields (max. 9)
54	GET/PUT field string lengths wrong
55	GET/PUT record number incorrect
60	Sprite string length incorrect
61	Error on reserve
62	Error in menu
63	Error in reserve
64	Error in pointer
90	Error in local
91	Error in for
92	Resume (Nex) impossible. Fatal, For or Local
100	Copyright 1986, GfA Systemtechnik

TOS Error messages

Error	Text
-1	General error
-2	Drive not ready. Time out
-3	Unknown command
-4	CRC error, Disk checksum wrong
-5	Bad request. Illegal command
-6	Seek error. Track not found
-7	Unknown media. Illegal boot sector
-8	Sector not found
-9	Out of paper
-10	Write error
-11	Read error
-12	General error 12
-13	Disk write-protected
-14	Disk was changed
-15	Unknown device
-16	Bad sector (verify)
-17	Insert next disk
-32	Illegal function number
-33	File not found
-34	Pathname not found
-35	Too many files open
-36	Access not allowed
-37	Illegal handle
-39	Memory full
-40	Illegal memory block address
-46	Illegal drive specifier
-49	No more data
-64	GemDOS area error. Seek incorrect?
-65	Internal GemDOS error
-66	Not a binary program file
-67	Memory block error

Bomb error messages

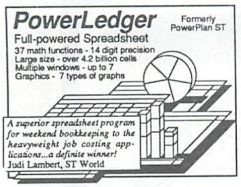
Error	Text
102	2 bombs - bus error. Perhaps incorrect PEEK or POKE
103	3 bombs - address error. Odd word address for DPOKE, DPEEK, LPOKE or LPEEK
104	4 bombs - illegal instruction. Execution of an invalid 68000 machine language command
105	5 bombs - divide by zero (in 68000 machine language)
106	6 bombs - CHK exception. 68000 interrupt by the CHK command
107	7 bombs - TRAPV exception. 68000 interrupt by the TRAPV command
108	8 bombs - PRIVILEGED violation. 68000 interrupt by execution of a PRIVILEGED command
109	9 bombs - TRACE exception - 68000 TRACE interrupt

Quick Index

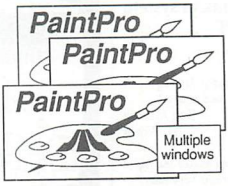
Subject Index

Index

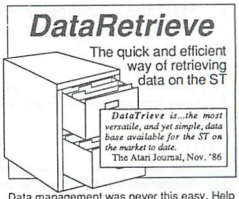
We have the software you've been looking for!



A powerful, large capacity (65,000 by 65,000 cells) spreadsheet package that also features a built-in calculator, online notepad and integrated graphics. Displays your data in numerical or graphical format instantly in up to seven different chart types (pie, bar, 3D bar, line, etc.). 14 digit precision with 37 math functions and more. **$79.95**



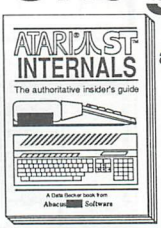
Express yourself with **PaintPro**, the GEM-based, full-page graphics design package. Multiple windows. Cut & paste between windows. Free-form sketching; lines, circles, boxes, text, fill, zoom, undo, rotate, invert, help. Edit fill and line patterns. **$49.95**
PaintPro Library #1–hundreds of clip art pictures, 5 GDOS fonts. Fills 3 disks. **$29.95**



Data management was never this easy. Help screens; lightning-fast operation; tailorable display; user-definable edit masks; up to 64,000 records. Supports multiple files. Includes RAM-disk programs. Complete search, sort and file subsetting. Interfaces to TextPro. Easy printer control. Includes five common database setups. **$49.95**

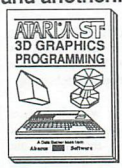
ST Disk Drives - Inside and Out
Includes chapters on files, file structures and data management. Thoroughly discusses floppy disks, hard disks and RAM disks from a programming and a technical perspective. Several full-length utilities and tools to further explore the ST disk drives. 450pp $24.95

One good book... deserves another...



ST INTERNALS
Essential and valuable information for the professional programmer and ST novice. Detailed descriptions of the sound and graphics chips, internal hardware, I/O ports, using GEM, system variables, interrupt instructions, error codes. Commented BIOS listing. An indispensible reference for your ST library. 450pp **$19.95**

and another...



3D Graphics Programming
FANTASTIC! Rotate about any axis, zoom in or out, and shade 3D objects. Programs written in machine language (commented) for high speed. Learn the theory behind 3D graphics; shading, hidden line removal. With 3D pattern maker & animator. **$24.95**

and another...

ST TRICKS & TIPS
Fantastic collection of programs and info for the ST. Complete programs include: super-fast RAM disk; time-saving printer spooler; color print hardcopy; plotter output hardcopy; creating accessories. Money saving tricks and tips. 260pp **$19.95**

and another...

GEM Programmer's Ref.
For serious programmers needing detailed information on GEM. Presented in an easy-to-understand format. All examples in C and assembly language. Covers VDI and AES functions. No serious programmer should be without. 410pp **$19.95**

Atari and Atari ST are trademarks of Atari Corp.
GEM is a trademark of Digital Research Inc.

Optional diskettes are available for $14.95 each.

you can count on **Abacus**

5370 52nd Street SE
Grand Rapids, MI 49508
Phone (616) 698-0330

Other software and books also available. Call or write for your **free catalog** or the name of your nearest dealer. Or you can order directly using your Visa, MC or Amex. Add $4.00 per order for shipping and handling. Foreign orders add $12.00 per item. 30-day money back guarantee on software. Dealers inquires welcome—over 2000 dealers nationwide.

Selected Abacus Products for the ATARI® ST™

AssemPro

Machine language development system for the Atari ST

"...I wish I had (AssemPro) a year and a half ago... it could have saved me hours and hours and hours."
—Kurt Madden
ST World

"The whole system is well designed and makes the rapid development of 68000 assembler programs very easy."
—Jeff Lewis
Input

AssemPro is a complete machine language development package for the Atari ST. It offers the user a single, comprehensive package for writing high speed ST programs in machine language, all at a very reasonable price.

AssemPro is completely GEM-based—this makes it easy to use. The powerful integrated editor is a breeze to use and even has helpful search, replace, block, upper/lower case conversion functions and user definable function keys. **AssemPro**'s extensive help menus summarizes hundreds of pages of reference material.

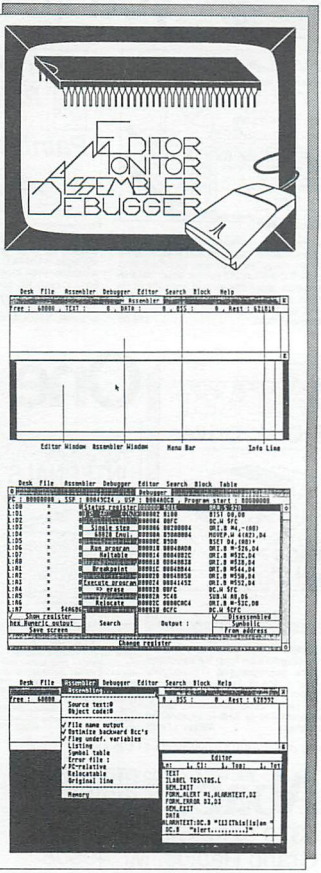
The fast macro assembler assembles object code to either disk or memory. If it finds an error, it lets you correct it (if possible) and continue. This feature alone can save the programmer countless hours of debugging.

The debugger is a pleasure to work with. It features single-step, breakpoint, disassembly, reassembly and 68020 emulation. It lets users thoroughly and conveniently test their programs immediately after assembly.

AssemPro Features:

- Full screen editor with dozens of powerful features
- Fast 68000 macro assembler assembles to disk or memory
- Powerful debugger with single-step, breakpoint, 68020 emulator, more
- Helpful tools such as disassembler and reassembler
- Includes comprehensive 175-page manual

AssemPro Suggested retail price: **$59.95**



Selected Abacus Products for the ATARI® 人ST™

BeckerText ST

The High-Powered Word Processing Package for the ST

A word processing package for serious Atari ST owners. Because **BeckerText** is more than a word processor.

It has all the features of our **TextPro**, and more: WYSIWYG formatting and printing, graphic merge capabilities, automatic hyphenation and indexing of your documents.

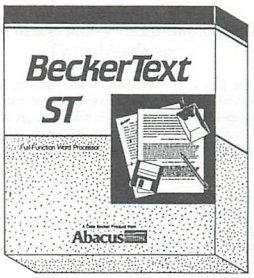
But **BeckerText** also does a few things that you might not expect...like calculate numbers within text, with templates for calculations in up to five columns. (It's just like having a spreadsheet program built into your word processor!). **BeckerText** prints up to five columns of text a page for professional-looking newsletters, presentations, reports, etc. It even has two expandable spelling checkers for 100% spelling accuracy.

BeckerText is also a perfect choice for C language programmers as an extremely flexible C editor. Whether you're deleting, adding or duplicating a block of C source code, **BeckerText** does it all, automatically. The online dictionary can double as a C syntax checker—catch those syntax errors immediately.

BeckerText gives you the power and flexibility to produce the professional-quality documents that you demand. It adapts to most popular dot-matrix and letter-quality printers. Includes a comprehensive tutorial, manual and glossary.

When you need more from your word processor than just word processing, you need **BeckerText**. Discover the power of **BeckerText**.

Suggested retail price: **$99.95**



BeckerText Features:

- Select options from dropdown menus or shortcut keys
- Fast WYSIWYG formatting
- Bold, italic, underline, superscript and subscript characters
- Automatic wordwrap and page numbering
- Sophisticated tab and indent options, with centering & margin justification
- Move, Copy, Delete, Search & Replace options
- Automatic hyphenation & automatic indexing
- Write up to 999 characters per line with horizontal scrolling feature
- Online dictionary checks spelling as you're writing
- Spelling checker interactively proofs text
- Calculates numbers within text—use templates to calculate in columns
- Customize up to 30 function keys to store often-used text and macro commands
- Merge graphics into documents
- Includes ***BTSnap*** program for converting text blocks to graphics
- C-source mode for quick and easy C language program editing
- Multiple-column printing—up to five columns on a single page
- Adapts to virtually any dot-matrix or letter-quality printer
- Load & save files through the RS-232 port
- Comprehensive tutorial and manual
- Not copy protected

Selected Abacus Products for the ATARI® 1040ST™

Chartpak ST

Professional-quality charts and graphs on the Atari ST

In the past few years, Roy Wainwright has earned a deserved reputation as a topnotch software author. **Chartpak ST** may well be his best work yet. **Chartpak ST** combines the features of his **Chartpak** programs for Commodore computers with the efficiency and power of GEM on the Atari ST.

Chartpak ST is a versatile package for the ST that lets the user make professional quality charts and graphs fast. Since it takes advantage of the ST's GEM functions, **Chartpak ST** combines speed and ease of use that was unimaginable til now.

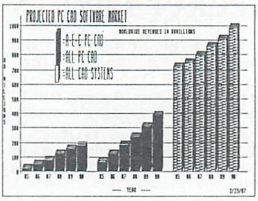
The user first inputs, saves and recalls his data using **Chartpak ST**'s menus, then defines the data positioning, scaling and labels. **Chartpak ST** also has routines for standard deviation, least squares and averaging if they are needed. Then, with a single command, your chart is drawn instantly in any of 8 different formats—and the user can change the format or resize it immediately to draw a different type of chart.

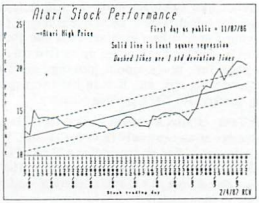
In addition to direct data input, **Chartpak ST** interfaces with ST spreadsheet programs spreadsheet programs (such as **PowerLedger ST**). Artwork can be imported from **PaintPro ST** or DEGAS. Hardcopy of the finshed graphic can be sent most dot-matrix printers. The results on both screen and paper are documents of truly professional quality.

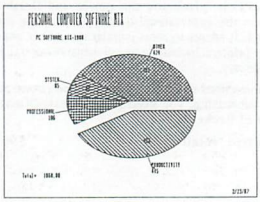
Your customers will be amazed by the versatile, powerful graphing and charting capabilities of **Chartpak ST** .

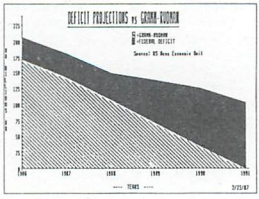
Chartpak ST works with Atari ST systems with one or more single- or double-sided disk drives. Works with either monochrome or color ST monitors. Works with most popular dot-matrix printers (optional).

Chartpak ST Suggested Retail Price: **$49.95**









Selected Abacus Products for the ATARI® ST™

TextPro

Wordprocessing package for the Atari ST

"TextPro seems to be well thought out, easy, flexible anf fast. The program makes excellent use of the GEM interface and provides lots of small enhancements to make your work go more easily... if you have an ST and haven't moved up to a GEM word processor, pick up this one and become a text pro."

—John Kintz
ANTIC

"TextPro is the best wordprocessor available for the ST"
—Randy McSorley
Pacus Report

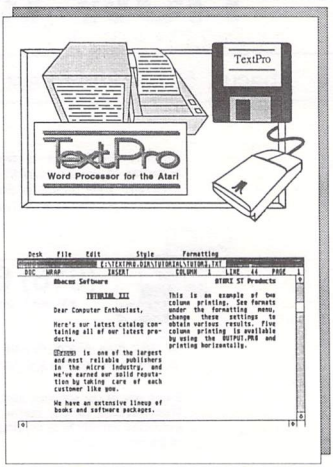
TextPro is a first-class word processor for the Atari ST that boasts dozens of features for the writer. It was designed by three writers to incorporate features that they wanted in a wordprocessor—the result is a superior package that suits the needs of all ST owners.

TextPro combines its "extra" features with easy operation, flexibility, and speed—but at a very reasonable price. The two-fingered typist will find **TextPro** to be a friendly, user-oriented program, with all the capabilities needed for fine writing and good-looking printouts. **Textpro** offers full-screen editing with mouse or keyboard shortcuts, as well as high-speed input, scrolling and editing. **TextPro** includes a number of easy to use formatting commands, fast and practical cursor positioning and multiple text styles.

Two of **TextPro**'s advanced features are automatic table of contents generation and index generation —capabilities usually found only on wordprocessing packages costing hundreds of dollars. **TextPro** can also print text horizontally (normal typewriter mode) or vertically (sideways). For that professional newsletter look, **TextPro** can print the text in columns—up to six columns per page in sideways mode.

The user can write form letters using the convenient Mail Merge option. **TextPro** also supports GEM-oriented fonts and type styles—text can be **bold**, underlined, *italic*, superscript, outlined, etc., and in a number of point sizes. **TextPro** even has advanced features for the programmer for development with its Non-document and C-sourcecode modes.

TextPro Suggested Retail Price: **$49.95**



TextPro ST Features:

- Full screen editing with either mouse or keyboard
- Automatic index generation
- Automatic table of contents generation
- Up to 30 user-defined function keys, max. 160 characters per key
- Lines up to 180 characters using horizontal scrolling
- Automatic hyphenation
- Automatic wordwrap
- Variable number of tab stops
- Multiple-column output (maximum 5 columns)
- Sideways printing on Epson FX and compatibles
- Performs mail merge and document chaining
- Flexible and adaptable printer driver
- Supports RS-232 file transfer (computer-to-computer transfer possible)
- Detailed 65+ page manual

TextPro works with Atari ST systems with one or more single- or double-sided disk drives. Works with either monochrome or color ST monitors.

TexPro allows for flexible printer configurations with most popular dot-matrix printers.

GFA BASIC

Quick Program Reference Guide for the Atari ST

GFA BASIC has become one of the most popular BASIC interpreters for the Atari ST. Its flexibility, speed, large command set, and structured coding capabilities make GFA BASIC a powerful tool for the ST programmer.

The GFA BASIC Quick Program Reference Guide is also a powerful tool for the GFA BASIC programmer. Its easy-to-read, easy-to-locate organization helps the programmer find commands, their uses and parameters in a flash. With its clear alphabetical listing, Quick Index and compact format, the GFA BASIC Quick Program Reference Guide is truly instant information at your fingertips.

ISBN 0-916439-90-9

Abacus

5370 52nd Street SE • Grand Rapids MI • 49508